FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

MASTER'S THESIS

OPTIMIZATION OF UTILIZATION OF TEST RESOURCES

Author	Kari Orava
Supervisor	Jussi Haapola
Second Examiner	Markku Juntti
Technical Advisor	Matti Kärki

September 2019

Orava K. (2019) Optimization of utilization of test resources. University of Oulu, Faculty of Information Technology and Electrical Engineering, Degree Programme in Electronics and Communications Engineering. Master's Thesis, 90 p.

ABSTRACT

Limited testing resources are one of the most fundamental challenges in testing. Testing of complex systems will require very large numbers of test cases to provide an adequate level of testing. Coverage is a popular metric to state the level of testing. However, coverage alone is not always a good measure to describe the level of testing for two reasons. First, it does not provide information of how efficiently the testing resources were spent. Second, coverage does not contain knowledge of how close to the optimal utilization the testing is. This thesis proposes a way to measure the level of test resource utilization, and a way to estimate the distance from the optimal resource utilization.

In this thesis a set of efficiency and performance metrics are defined to measure utilization of testing resources. The defined metrics consider the achieved coverage with respect to spent testing resources and the complexity of the tested system. Based on the defined metrics, an approximation formula for the maximum efficiency as a function of available testing resources is defined. A method to simplify complex equations by considering the states of equation is proposed. The defined metrics and proposed method are applied into a 3GPP equation, intended for a Long Term Evolution (LTE) device, to search a subset that maximizes the test resource utilization.

The optimization of the utilization of test resources is viewed as a set cover problem, which is attempted so solve with various algorithms, such as brute force algorithm, classical Greedy Algorithm (GA), and a few of their variants and combinations. Performance of the algorithms are studied and compared. Performance results are presented, and the best results compared with the approximated maximum.

It was observed that there was not a single algorithm that suits for all scenarios, but the choice of algorithms depends on the resources available. Brute force-based algorithms should be selected when there are scarce resources, and GA-based algorithms when resources are plentiful. Based on the results, the utilization of the test resources was maximized with a moderate number of test resources.

Key words: Testing, LTE, Optimization, Algorithm

Orava K. (2019) Testiresurssien käytön optimointi. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Diplomityö, 90 s.

TIIVISTELMÄ

Rajalliset testausresurssit ovat yksi keskeisimmistä haasteista testauksessa. Monimutkaisten järjestelmien testaus tarkoittaa usein todella suurta määrää testejä, jotta saavutettaisiin riittävä testauksen taso. Kattavuus on perinteinen tapa mitata testauksen tasoa. Kattavuus ilmaisee absoluuttisen testauksen tason testattujen ja testaamattomien osioiden suhteena. Kattavuus yksin ei ole paras tapa kuvata testauksen tasoa kahdesta syystä. Kattavuus ei ilmaise kuinka tehokkaasti testaus resurssit käytettiin. Kattavuus ei myöskään kerro kuinka lähellä optimaalista testaus resurssien käyttöä oltiin. Tässä diplomi työssä esitetään vaihtoehtoinen tapa mitata testauksen tasoa, sekä keinon arvioida, kuinka lähellä ollaan optimaalista testausta.

Tässä työssä määritellään joukko metriikoita, joilla mitataan kuinka tehokkaasti testausresurssit käytetään hyödyksi. Metriikat ottavat huomioon saavutetun kattavuuden suhteessa käytettyihin resursseihin sekä testattavan järjestelmän monimutkaisuuden. Määriteltyjen metriikoiden pohjalta määritellään approksimaatiokaava, joka ilmaisee suurimman mahdollisen hyötysuhteen resurssien määrän funktiona. Menetelmä monimutkaisten yhtälöiden yksinkertaistamiseen käsittelemällä yhtälön tiloja ehdotetaan. Määriteltyjä metriikoita sekä ehdotettua menetelmää sovelletaan Long Term Evolution (LTE) laitteelle tarkoitettuun 3GPP kaavaan, ja pyritään löytämään testijoukko, joka optimoi testausresurssien käytön.

Testausresurssien optimointia käsitellään joukko kattavuus ongelmana, jota yritetään ratkaista useilla algoritmeilla, kuten raaka voima haku algoritmilla sekä ahneella algoritmilla, sekä muutamalla näiden kahden variaatiolla ja yhdistelmällä. Algoritmien tulokset esitetään ja niitä vertaillaan. Parhaita tuloksia verrataan approksimoituun maksimitehokkuuteen.

Työssä havaitaan että yksi algoritmi ei sovellu joka tilanteeseen, vaan paras algoritmi riippuu käytettävissä olevien resurssien määrästä. Raaka voima algoritmi saavuttaa parhaan tuloksen pienille resurssimäärille, kun taas ahne algoritmi suurille. Tulosten perusteella paras testausresurssien hyödyntäminen saavutetaan kohtalaisella resurssimäärällä.

Avainsanat: Testaus, LTE, Optimointi, Algoritmi

TABLE OF CONTENTS

FOREWORD

This Master's Thesis was done for Nordic Semiconductor in Oulu. The initial idea for the thesis came from Matti Kärki, which served as inspiration for final subject's selection.

Subject was very interesting as it could be, at most part, integrated to my every day work, and hopefully benefit the whole company. During the thesis, I've gained some valuable insight in algorithm implementation and coding with Python.

I would like to thank my technical advisor Matti Kärki for the initial idea for the thesis, for weekly meetings that were very useful as a motivational tool, and a place where I got to verbally recap what had been done, which often pointed the next steps and potential deficiencies of the thesis. My supervisor Jussi Haapola for patiently going through the ever-increasing thesis document, and the clear, detailed commenting.

Also, I would like to thank Pauliina for constant support and for acting as a test subject to test the comprehensibility of the text. And my family for being completely clueless about the subject so that I could feel smart for once.

Oulu, May, 5 2019

Kari Orava

LIST OF ABBREVIATIONS AND SYMBOLS

3GPP	Third Generation Partnership Project
BGA	Big step Greedy Algorithm
BL	Bandwidth Limited
BS	Base Station
BW	Bandwidth
CBE	Coverage Based Efficiency
CE	Coverage Enhanced
C-NIC	Combinatorial Non-overlapping Input Coverage
CC	Combinatorial Coverage
CS	Combination Space
DL	Downlink
eNB	Evolved Node B
GA	Greedy Algorithm
IIC	Individual Input Coverage
LTE	Long Term Evolution
LTE-A	Long Term Evolution – Advanced
LTE-M	Long Term Evolution for Machine Type Communications
MPDCCH	Physical Downlink Control Channel for Machine Type Communications
MTC	Machine Type Communications
NB	Narrowband
NB-IoT	Narrowband Internet of Things
NIC	Non-overlapping Input Coverage
OC	Output Coverage
OFDM	Orthogonal Frequency Division Multiplexing
PDSCH	Physical Downlink Shared Channel
PRACH	Physical Random Access Channel
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
RAN	Radio Access Network
RBE	Result Based Efficiency
SGD	Sub-Group Division
R-SGD	Recursive Sub-Group Division
TC	Test Case
TRU	Test Resource Unit
TS	Technical Specification
UE	User Equipment
UL	Uplink
WGA	Weighed Greedy Algorithm

B	Optimal Example Subset of size four
C_{cnic}	Combinatorial Non-overlapping Input Coverage
C_{iic}	Individual Input Coverage
$C_{\mathrm{iic},n}$	Individual Input Coverage of n^{th} Parameter
$C_{\mathrm{iic},S}$	Individual Input Coverage of subset S
$C_{\mathrm{inc,oc}}$	Overall Coverage Increment per each included Output Value

$C_{\mathrm{inc,cnic}}$	Overall Coverage Increment per each included Combinatorial Non-overlapping Input Coverage Combination
$C_{\mathrm{inc},i}$	Overall Coverage Increment per each Included Value of i^{th} Parameter
C_{nic}	Non-overlapping Input Coverage
$C_{\mathrm{max,single}}$	Maximum Possible Coverage Increment for a Single Combination
C_{oc}	Output Coverage
C_{overall}	Overall Coverage
$C_{\mathrm{term},k}$	Term Coverage of k^{th} Equation Term.
E	Example Subset including all the Combinations
$f_{\mathrm{NB,hop}}^{\mathrm{DL}}$	Narrowband Offset between one Narrowband and the next Narrowband
h	Hardness Factor
i_0	Absolute Subframe Number of the First Downlink Subframe intended for first Physical Downlink Shared Channel Transmission.
$N_{\mathrm{RB}}^{\mathrm{DL}}$	Downlink Bandwidth Configuration, expressed in multiples of $N_{\mathrm{sc}}^{\mathrm{RB}}$
$N_{\mathrm{sc}}^{\mathrm{RB}}$	Resource Block Size in the Frequency Domain, expressed as a number of Subcarriers
$n_{\mathrm{NB}}^{(i_0)}$	Narrowband of First Absolute Subframe Intended for Physical Downlink Shared Channel Transmission
$N_{\mathrm{all},n}$	Number of Different Possible Values of n^{th} Parameter
N_{C}	Number of Possible ways to select R_{min} Combinations from N_{cc} Combinations
N_{cc}	Number of all Possible Combinatorial Coverage Combinations for an Equation
$N_{\mathrm{NB}}^{\mathrm{ch,DL}}$	Number of Consecutive Absolute Subframes over which MPDCCH or PDSCH stays at the same Narrowband before hopping to another Narrowband, expressed as Number of Absolute Subframes
$N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$	Number of Narrowbands over which MPDCCH or PDSCH Frequency Hops expressed as Number of Downlink Narrowbands
N_{cnic}	Number of Different Possible Combinatorial Non-overlapping Input Coverage Combinations
N_{comp}	Number of Comparisons
$N_{\mathrm{comp,sub}}$	Number of Comparisons for a Sub-group
$N_{\mathrm{symb}}^{\mathrm{DL}}$	Number of Orthogonal Frequency Division Multiplexing Symbols in a Downlink Slot
$N_{\mathrm{NB}}^{\mathrm{DL}}$	Number of Downlink Narrowbands
N_{oc}	Number of Different Possible Output Values for an Equation
$N_{\mathrm{inc,cc}}$	Number of Included Combinatorial Coverage Combinations
$N_{\mathrm{inc,cnic}}$	Number of Included Combinatorial Non-overlapping Input Coverage Combinations
$N_{\mathrm{inc},i}$	Number of Included Values of i^{th} Parameter
$N_{\mathrm{inc,oc}}$	Number of Included Output Values
N_{par}	Number of Parameters
$N_{\mathrm{abs}}^{\mathrm{PDSCH}}$	Total Number of Absolute Subframes over which Physical Downlink Shared Channel with Repetition spans
$N_{\mathrm{rep}}^{\mathrm{PDSCH}}$	Number of Physical Downlink Shared Channel Repetitions, expressed as a Number of Absolute Subframes

$N_{\text{states,eq}}$	Number of Included Equation States
$N_{\text{states,eq,all}}$	Number of Different Possible Equation States
N_{sub}	Number of Sub-groups
N_{term}	Number of Equation Terms
$N_{\text{term,state},k}$	Number of Included Equation Term States of k^{th} Term
$N_{\text{term,state,all},k}$	Number of Different Possible Term States of k^{th} Term
N_{val}	Number of Different Parameter Values
$N_{\text{val},i}$	Number of Different Possible Values of i^{th} Parameter
O	An Output Value
P_{cbe}	Absolute Subset Performance
$P_{\text{cbe,r}}$	Relative Subset Performance
R	Number of Resources expressed in Test Resource Units
R_{min}	Minimum Number of Required Resources to achieve Full Overall Coverage
$R_{\text{min,cnic}}$	Minimum Number of Required Resources to achieve Full Combinatorial Non-overlapping Input Coverage
$R_{\text{min,iic}}$	Minimum Number of Required Resources to achieve Full Individual Input Coverage
$R_{\text{min,oc}}$	Minimum Number of Required Resources to achieve Full Output Coverage
$R_{\text{min,par},i}$	Minimum Number of Required Resources to achieve Full Parameter Coverage for i^{th} Parameter
S	Sub-optimal Example Subset of Size Four
T_{slot}	Slot Duration
Q	Example Optimal Full Coverage Subset
ε_{cbe}	Coverage Based Efficiency
ε_{cc}	Coverage Based Efficiency with respect to Combinatorial Coverage
$\varepsilon_{\text{cnic}}$	Coverage Based Efficiency with respect to Combinatorial Non-overlapping Input Coverage
ε_{iic}	Coverage Based Efficiency with respect to Individual Input Coverage
ε_{nic}	Coverage Based Efficiency with respect to Non-overlapping Input Coverage
ε_{oc}	Coverage Based Efficiency with respect to Output Coverage
ε_{opt}	Optimal Overall Efficiency
$\varepsilon_{\text{opt,full}}$	Optimal Full Coverage Efficiency
$\varepsilon_{\text{opt,single}}$	Optimal Discrete Efficiency
$\varepsilon_{\text{opt,single,max}}$	Optimal Discrete Efficiency of a Coverage Metric that requires Least Number of Combinations for Full Coverage
$\varepsilon_{\text{overall}}$	Coverage Based Efficiency with respect to Overall Coverage
ε_{r}	Relative Efficiency
θ	Maximum Subset Size that achieves Full Optimal Discrete Efficiency
$\mu_{\text{out},n}$	Average Number of Output Values included in n^{th} Sub-group during the optimization
$\mu_{\text{cnic},n}$	Average Number of Combinatorial Non-overlapping Input Coverage Combinations included in n^{th} Sub-group during the optimization

1 INTRODUCTION

Restricted testing resources are one of the constraints in software testing. When the system under testing becomes complex, testing everything might not be feasible, especially, if limited time is available [1]. Coverage is a one common metric to measure the level of testing [2]. Coverage describes the absolute level of testing, i.e. the ratio of the tested and untested sections of the tested system. Full coverage, or as high coverage as possible, is desired to provide reliability for the tested system. However, if there are limited resources for testing, coverage alone is not the best way of measuring the level of testing for two reasons. Firstly, coverage does not tell how well the spent resources were utilized. Secondly, because full coverage might be impossible to achieve with limited resources, a way to estimate the distance to optimal resource allocation is desired, as coverage does not indicate whether the achieved coverage was the highest possible for spent resources. Instead, a way to measure the relative performance of testing is desired to give a better estimation of the efficiency of the utilization of the testing resources. The purpose of this thesis is to find a metric for test resource utilization, and a way to estimate the distance to an optimal subset.

Long Term Evolution for Machine Type Communications (LTE-M) is chosen as a context technology because LTE-M is widely deployed technology [3] Given that there exists an LTE-M infrastructure, many new applications and devices utilizing the infrastructure will likely be developed in the future. New devices will require a comprehensive testing before they can utilize the infrastructure. LTE-M is also a relatively complex technology as its functionality is determined by a very large number of parameters, LTE-M provides a good challenge for testing and is therefore a good target for this thesis.

LTE-M is a sub-technology of the LTE. LTE is standardized by e.g. Third Generation Partnership Project (3GPP), that provides technical specifications for the LTE functionality. Scope of this thesis is restricted in physical layer functionality because of the large number of parameters present, the large number of parameters results that there are a very large number of parameter combinations to test, which further encourages to determine optimal way to utilize testing resources. More precisely, focus in this thesis is on a single equation intended for a procedure that a Bandwidth Limited, Coverage Enhanced (BL/CE) device must be able to carry out.

To tackle the large number of input combinations, a Combinatorial Non-overlapping Input Coverage (C-NIC) decomposition is performed for the equation. C-NIC decomposition produces so called C-NIC combinations, that contain the information of the state of equation for respective input stimulus. C-NIC decomposition is a useful tool because going through all C-NIC combinations ultimately means that each state of an equation is considered. Therefore, including all C-NIC combinations instead of all input combinations mitigates the complexity of the equation. C-NIC combinations in conjunction with the output and input values are used to form coverage metric denoted as overall coverage, which aims to measure the level of the absolute level of testing with respect to system inputs, states, and outputs.

Optimization in this thesis is more algorithm-oriented than it is mathematical. Optimization is started from a brute force algorithm that searches the best solution by going through all the possibilities and selecting the best solution. However, it is obvious that such approach does not work for complex systems. Consequently, an approximation algorithm, Sub Group Division (SGD), is proposed. SGD finds a solution in feasible time with a cost of potentially losing the optimality. In addition, performance of a classical greedy algorithm (GA) is compared to the SGD. Further, the performance of various variants and combinations of the GA and SGD algorithms are compared.

The performances of subsets are evaluated with metrics such as: optimal efficiency approximation, achieved overall coverage, and hardness factor *h*. The hardness factor describes the complexity of the evaluated system. Hardness factor is defined as the number of possible combinations to select a full coverage subset for the evaluated system. Three quantities to evaluate the subset performance are introduced: absolute performance, relative performance, and relative efficiency. Absolute performance is a scalar value that can be used to find an optimal subset size and compare the resource utilization of different subsets. Relative performance denotes the distance to an optimal subset with full coverage. Relative efficiency describes the distance to optimal subset for an arbitrary sized subset. Feasibility of C-NIC decomposition, and the introduced metrics are tested on a real-life equation from a technical specification, targeted for a LTE device.

2 TERMINOLOGY

This chapter introduces the core terminology regarding this thesis. Terminology is divided into two sections: context-specific- and thesis-specific terminology. Context-specific terminology contains terminology regarding the subject context around which this thesis is written. Thesis-specific terminology addresses terminology that can have multiple meanings outside the thesis. The purpose of the thesis-specific terminology section is to further specify the meaning of most frequently used expressions throughout the thesis.

2.1 Context-specific terminology

Context specific terminology introduces concepts such as: cellular network, LTE, 3rd Generation Partnership Project (3GPP), Release, specification, uplink, downlink, and physical wireless channel. First, an overview, and the basic principle of cellular system is presented with a respective protocol, LTE. Second, the purpose of standardization organization 3GPP is explained along with related terms: release, and specification. Finally, the concept of physical channel explained along with example channels: Physical Downlink Shared Channel (PDSCH) and Physical Uplink Control Channel (PUCCH).

2.1.1 Cellular Network

A cellular network in this context means an infrastructure for wireless communication. The cellular network consists of Base Stations (BS) and potentially mobile users. A single cell in the cellular network resembles the coverage area of a single BS, i.e. the area in which the BS can operate. The cellular network provides coverage to the users within the range of any of its cells. Cellular networks are primary designed for multiuser scenario, in which users are interested in specific messages and can communicate with the network. [4]
LTE is a cellular network technology in which the BS is denoted as evolved Node B (eNB), and a user as User Equipment (UE). Radio Access Network (RAN) in LTE was primarily designed for full-duplex operation in paired spectrum, i.e. separate spectrum bands for uplink (UL) and downlink (DL) [5]. UL denotes the communication from UE to eNB, and DL the communication from eNB to UE. Full-duplex operation means that there can be simultaneous communication in both UL and DL. Also, the LTE supports several bandwidths (BW). The supported BWs are: 1.4, 3, 5, 10, 15, and 20 MHz [5]. A cellular network is illustrated in Figure 1.

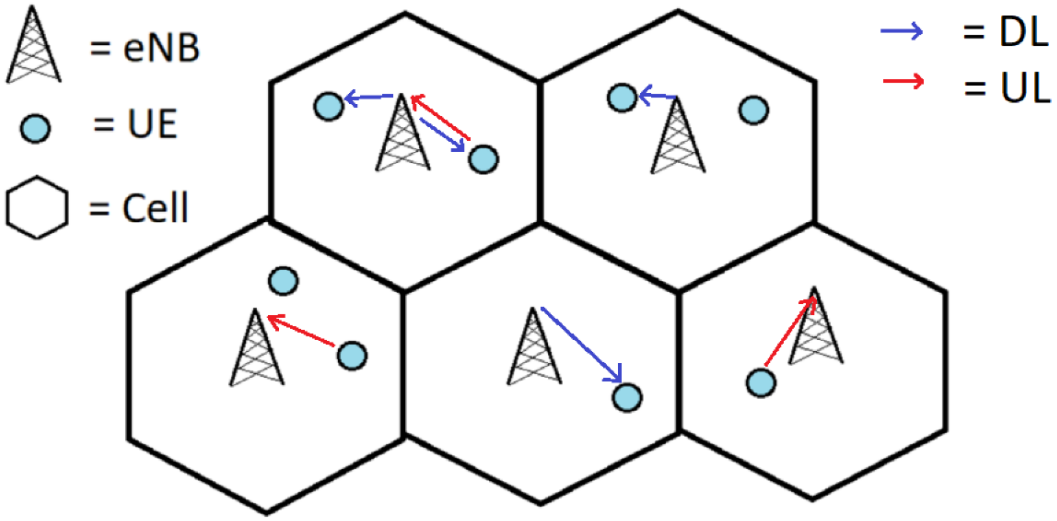

Figure 1. A cellular network.

2.1.2 Cellular Standard

A "standard", by definition, is "Something used as a measure, norm, or model in comparative evaluations" [6]. Cellular networks also have a standard which describes what is the norm for the system, i.e. what is expected from the system. Standards are provided by standardization organizations, such as 3GPP, which is one of the significant standardization organizations for cellular telecommunications [7]. In this context, standard can be viewed as a set of specifications for cellular telecommunication network.
A specification, by definition, is "An act of identifying something precisely or of stating a precise requirement." [8]. Further, specifications can be viewed as a set of rules that serve as instructions, e.g. for designers of devices, intended to be used for wireless transmission. If a device wishes to use an existing radio accessing scheme, e.g. cellular network, it must fulfil the specification of the mentioned scheme. Requiring devices to follow the specifications ensures that the devices, using same resource, e.g. spectrum band, can co-exists and their transmission can be orchestrated in such way that there is no interference or overlapping among their transmissions. Another purpose of the specifications is to allow interconnectivity between devices from different device manufacturers, utilizing the same access technology. For example, company A and company B both design and manufacture devices that utilize the same radio access technology. Specifications ensure that the devices, produced by both companies, work similarly, allowing company A's device to communicate with company B's device, although their implementations might differ.

The 3GPP provides specifications in form of a parallel releases. Each release is built on previous releases and consist of improvements for the existing technologies or entirely new functionalities. LTE was first introduced in 3GPP Release 8. LTE is the central radio access technology in modern wireless communications and many technologies in later releases are based on LTE. LTE-based technologies introduced per 3GPP release are: LTE-Advanced (LTE-A) in release 10, LTE for Machine-Type Communications (MTC) in release 13, and Narrowband Internet of Things (NB-IoT) in release 13. [7]

The 3GPP standards for LTE are provided in Technical Specification (TS) series 36. 3GPP uses numbering format 36.NNN-abc, where NNN is the number of the specification within the

series 36, a is the number of the release, b is the technical version number within the release, and c is an editorial version number used to track other than technical changes. [9]

2.1.3 Physical Channel

A "Channel" in this context means a medium over which an information signal is transmitted. "Physical" in this context means that the channel is non-abstract, i.e. there is a realization for the channel. For example, there is a physical channel between two antennas transmitting to each other over air interface.
The 3GPP has defined several physical channels for the LTE, used for various physical layer procedures, such as: cell search, power control, and random-access procedures. There are separate channels for the UL and DL. The 3GPP specifies five physical uplink channels, including: Physical Uplink Shared Channel (PUSCH), Physical Uplink Control Channel (PUCCH), and Physical Random-Access Channel (PRACH) [10 page 17]. For the DL, the 3GPP specifies twelve physical channels, including: Physical Downlink Shared Channel (PDSCH), Physical Broadcast Channel (PBCH), and MTC Physical Downlink Control Channel (MPDCCH) [10 pp. 8-10]. Focus in this thesis remains in DL side.
LTE utilizes an Orthogonal Frequency Division Multiplexing (OFDM) as multiple accessing scheme in DL [10 pp. 8-10]. In the OFDM, the transmission is divided into orthogonal subcarriers in the frequency domain. OFDM subcarrier signal sidelobes are such that the carrier nulls are located on the peaks of the adjacent subcarriers, i.e. there are no interference [11].
In LTE, the resources are mapped in time and frequency domain. Mapping depends on the used frame structure type. In this thesis, frame structure type 1 is assumed. Frame structure type 1 is applicable for Frequency Division Duplexing (FDD). FDD means that the UL and DL utilize a different frequency bands for the transmissions. In frame structure type 1 the time domain is divided into radio frames, subframes, slots, and symbols. A radio frame is a 10 ms time interval, which is further divided into a ten subframes of 1 ms interval. A subframe is divided into two slots of 500 µs interval. [12 page 14] DL slots are divided into $N_{\mathrm{symb}}^{\mathrm{DL}}$ OFDM symbols, number of which is determined by cyclic prefix and sub-carrier spacing configuration [12 page 70].
In LTE, the frequency domain is divided into narrowbands (NB), Physical Resource Blocks (PRBs), and sub-carriers. An NB consists of 6 consecutive PRBs [12 page 73]. A PRB consists of $N_{\mathrm{sc}}^{\mathrm{RB}}$ consecutive subcarriers in frequency domain and N_{symb}^{DL} consecutive symbols in time domain. A sub-carrier is a Δf section of bandwidth, size of which is determined by the subcarrier spacing configuration [12 page 70]. In this thesis, normal cyclic prefix configuration and subcarrier spacing of 15 kHz are assumed. Normal cyclic prefix with $\Delta f = 15$ kHz corresponds to 7 symbols per slot, i.e. the symbol interval is roughly 71 µs [12 page 70].
The 3GPP identifies the physical channel as a set of resource elements that carry information, originating from upper layers. The resource elements are mapped into a resource grid via time index *l*, and frequency index *k*. Index *l* denotes a symbol index in a single slot, and *k* denotes an index of a subcarrier within system BW. A single resource element corresponds to a time-frequency index pair (*k*, *l*) in the resource grid. Figure 2, illustrates the resource grid for LTE DL, as specified in 3GPP. Table 1 presents explanations for symbols in Figure 2. [12 pp. 67-69]

Table 1. Explanations of symbols for Figure 2 [12 pp. 10-13]

Symbol	Explanation
$N_{\mathrm{RB}}^{\mathrm{DL}}$	Downlink BW configuration, expressed in multiples of N_{sc}^{RB}
$N_{\mathrm{sc}}^{\mathrm{RB}}$	Resource block size in the frequency domain, expressed as a number of subcarriers
$N_{\mathrm{symb}}^{\mathrm{DL}}$	Number of OFDM symbols in a downlink slot
T_{slot}	Slot duration

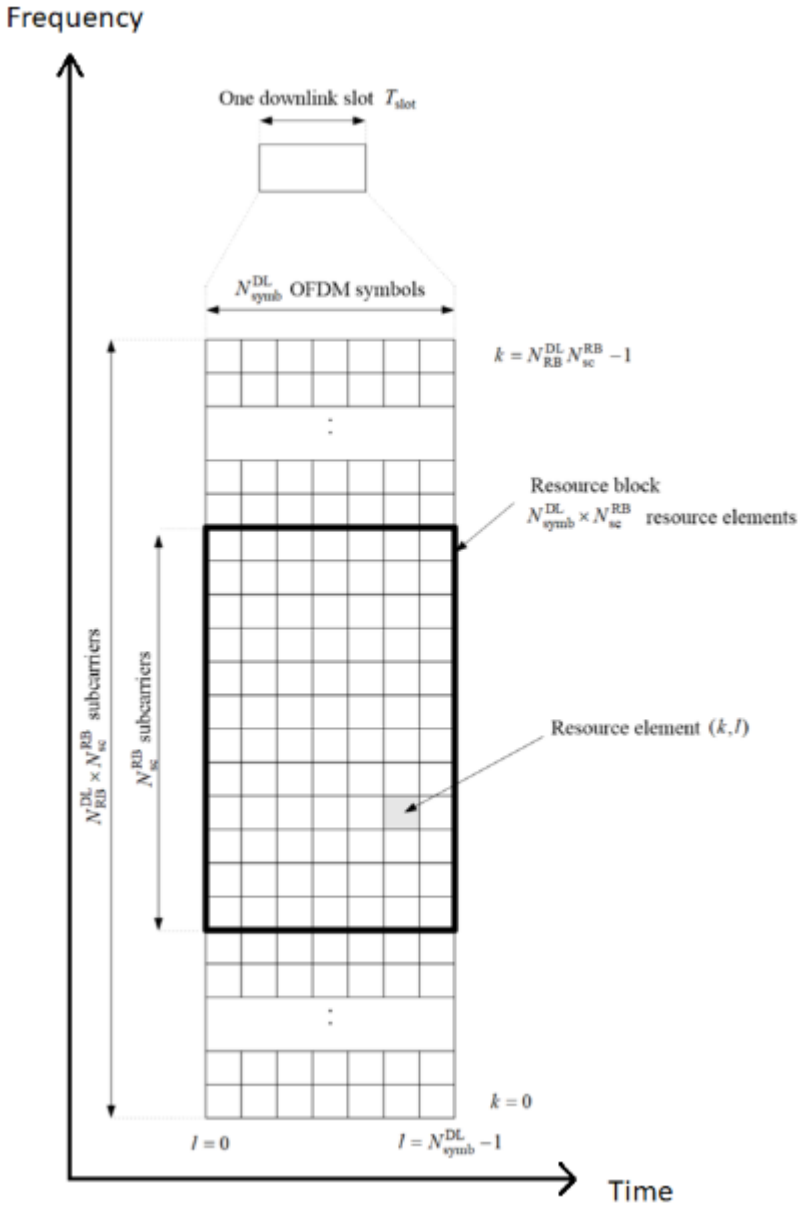

Figure 2. LTE DL resource grid.

2.2 Thesis-specific terminology

Thesis-specific terminology introduces the most central terminology regarding this thesis and specifies what is meant with terms having multiple meanings outside this thesis. The specified terminology includes: combination, parameter, output, subset, coverage, and combination space. First, the meaning of "combination" is specified and its relation to "parameter", and "output" is explained. Second, the concept of "subset" is introduced and its relation to "combination" is explained. Finally, larger scale concepts: "coverage" and "combination space" are explained and how they are related.

2.2.1 Parameter, input, output and combination

A parameter means a single system variable. A system variable is a value from a specified set of values that define a single aspect of system operation. A system can have multiple parameters depending on the number of the controllable variables, i.e. the complexity of the system. A state where all the system parameters have obtained a valid value is called as a system input, or an input combination. The system input determines the overall behaviour of the system. In this thesis, all the parameters are 3GPP specified parameters and the system is an equation defining a physical layer operation for DL NB hopping.

An output means the result after an input combination was inserted to the system. The output can be many things, including: a scalar value, a vector, a pattern, or a system state, depending what the system does. The numbers of the different outputs of a system depends on the numbers of the system variables or the possible input combinations. Outputs from different inputs can be different or equal, but each valid input combination always results in an output. In this thesis, an output means an equation result after an input is inserted to it.

A combination, as mentioned before, can denote a combined state of all the input parameters. Also, the combination can mean a combined state of any of the individual states of a system. For example, an equation can be divided into sections and each section will obtain an individual state after an input combination is inserted into the equation. In this thesis, a combination means either a combined state of the input parameters, or a combined state of an equation sections.

The Figure 3 illustrates the terminology presented in this section via an example system. The example system is controlled with three parameters: *p1*, *p2*, and *p3*. The example system performs the function $f(p1,\ p2,\ p3) = \left[(p1 + p2)^{(p1+p3)}\right] \mathrm{mod}(p2 + p3)$, which is divided into three sections: $S1 = (p1 + p2),\ S2 = (p1 + p3)$, and $S3 = (p2 + p3)$. The combined state of the sections *S1*, *S2*, and *S3* forms a system state combination. The output of the system is a scalar value.

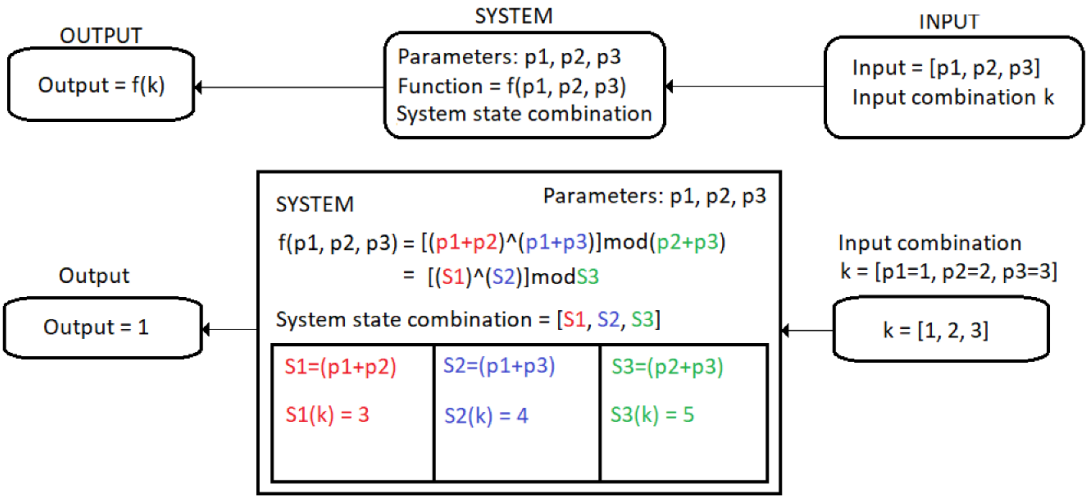

Figure 3. Example system.

In Figure 3, the example system is controlled with three input parameters: *p1*, *p2*, and *p3*. These three parameters form a single input combination, $k = [1, 2, 3]$, which drives the system into a specific state, based on the function $f(k)$. The system state consists of three sections: *S1*, *S2*, and *S3*, denoted with red, blue, and green. System states determine the output of the system. When k is introduced to the example system, *S1* becomes $p1+p2 = 3$, *S2* becomes $p1+p3 = 4$, and *S3* becomes $p2+p3 = 5$. Respective output, denoted as O, is obtained as illustrated in equation (2-1).

$$O = (S1^{S2})\mathrm{mod}(S3) = (3^4)\mathrm{mod}(5) = 81\mathrm{mod}5 = 1 \qquad (2-1)$$

2.2.2 Combination space, subset, and coverage

Combination space (CS) is a group of combinations. The CS can include all the possible combinations or a group of combinations that fulfil a specific requirement or criteria. In this thesis, CS means the combination group from which the combinations are selected when an optimal subset is searched. There can be more than one CS for a single system. In this thesis, overlapping between the CSs is not allowed, i.e. multiple different CSs must not contain the same combination. Although, a CS can be a part of a larger CS if the larger CS is not used in conjunction with the smaller CS, as this would cause overlapping in the overall CS.

In this thesis, a subset is a group of combinations, selected from a CS. A subset can be selected from a single CS or from multiple different CSs. All the elements in a single subset must be found from one of the system's CSs. In this thesis, it is assumed that a single subset can only contain unique combinations, i.e. multiple entries of any combination is not allowed within a single subset. The Figure 4 illustrates the relation between a CS and a subset.

In this thesis, a coverage means ratio of the numbers of the different occurrences of interest included in a subset to the numbers of all possible occurrences of interest found within a CS. The subject of interest can be e.g. parameter values, outputs, equation states, etc. Coverage is discussed more in Section 4.2.

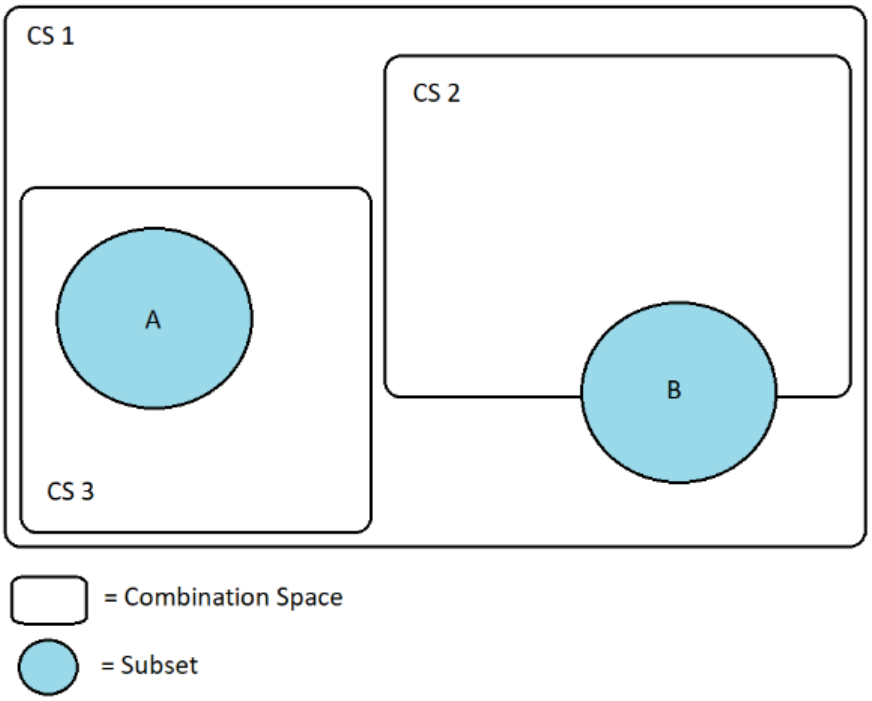

Figure 4. Relation between CS and subset within the scope of the thesis.

3 OPTIMIZATION

In this chapter, a short review of optimization is presented. First, the meaning of optimization is defined with help of an example problem. Second, various types of optimization problems are presented. Finally, a set cover problem is considered in detail as the optimization in this thesis can be viewed as a set cover problem.

3.1 Introduction

To define optimization, a term "optimum" must be defined. "Optimum" is Latin, meaning: "the ultimate ideal". Optimization can be interpreted as: "doing something as well as possible". When a system is being optimized, one attempts to steer the system towards its "optimum" or optimal state. The system is said to be optimal when it cannot perform any better. [13]

To optimize something, formulation of the optimization problem is required. An optimization problem usually consists of three parts: variables, objective function, and constraints [13]. Variables are the controllable sections of the problem, by changing the variables, the outcome of the objective function changes. The objective function models the system's behaviour in which the optimization takes place. The constraints are limitations or conditions that describe e.g. the feasibility or possibility of the system. An example of a constraint can be a non-negativity condition for mass in a real-life optimization problem.

As an example, consider two functions $f(x) = -x^2 + 2$ and $g(x) = x$. If one wishes to find the largest value of function $f(x)$ that is not larger than $g(x)$, the optimization problem for such problem is modelled in the equation (3-1).

$$\max_x f(x) \tag{3-1}$$

Subject to

$$f(x) \leq g(x)$$

In this problem, $f(x)$ is the objective, x the variable, and $f(x) \leq g(x)$ the constraint. First, it must be considered whether this problem is solvable. Given that $f(x)$ can achieve smaller and larger values than $g(x)$. E.g. $f(x = 3) < g(x = 3)$ and $f(x = 0) > g(x = 0)$, functions intersect at some point (*). In this case $g(x)$ restricts $f(x)$ from above as $f(x)$ achieves max value at x=0, which cannot be selected due to constraint condition (**). Because $f''(x) = -2$, $f(x)$ is twice differentiable, thus, $f(x)$ is concave (***) [14]. If a concave function is upper bounded by a linear function $g(x)$, the maxima is found from the intersection of $f(x)$ and $g(x)$. Problem reduces to quadratic equation:

$$\max_x\big(f(x) = g(x)\big) \tag{3-2}$$
$$\max_x(-x^2 + 2 = x) \tag{3-3}$$
$$\max_x(-x^2 - x + 2 = 0) \tag{3-4}$$
$$\max_x(-2, 1) = 1 \tag{3-5}$$

Solution to the optimization problem is x=1 and $f(1) = 1$. Figure 5 presents a graphical illustration of problem and the optimal point.

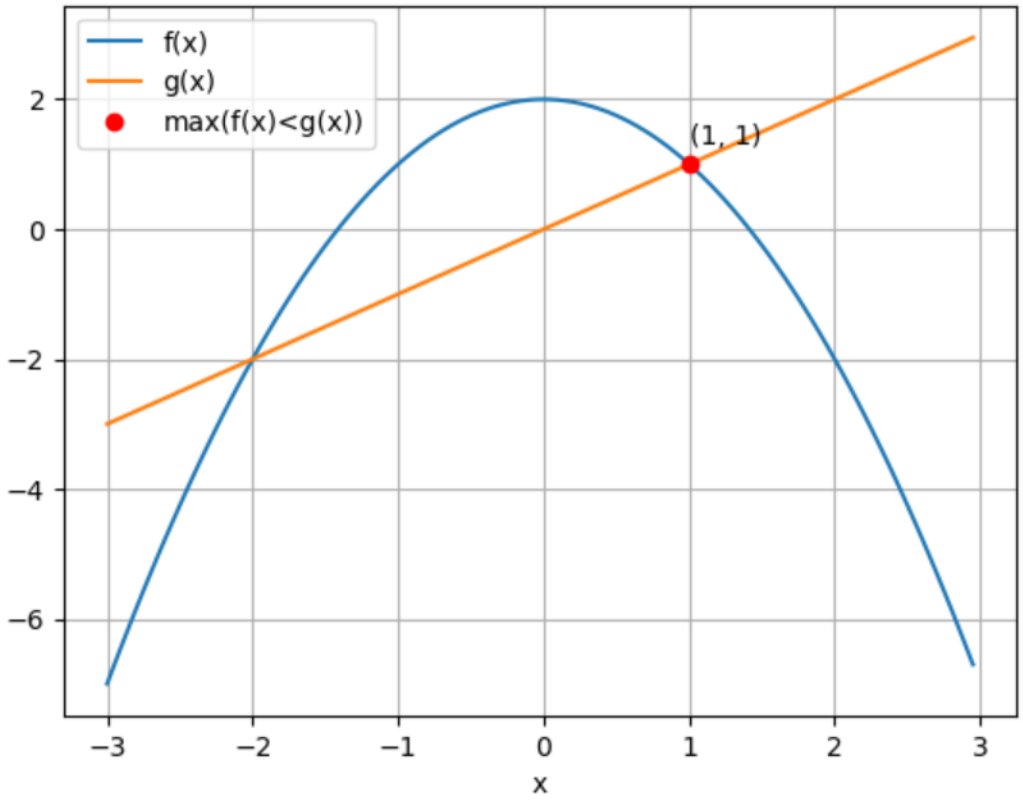

Figure 5. Graphical illustration of the optimization problem and the solution.

3.2 Set cover problem

A set cover problem is an optimization problem consisting of a universe U and subsets X_s of the universe U. The goal in the set cover problem is to find the minimum number of subsets X_s whose union equals the universe U. For example, consider a trivial set cover problem, where the universe U = {1, 2, 3, 4, 5, 6} and subsets X_1 = {1, 3, 5}, X_2 = {2, 4, 6}, and X_3 = {1, 2, 3}. Optimal solution to set cover problem is the minimum number of subsets that contain all the elements in U. In the example case, the optimal solution is achieved by selecting subsets X_1 and X_2, as their union equals the universe U, i.e. (X_1 ∪) = U. The set cover problem is known to be an NP-hard problem. [15 pp. 901-902]

NP stands for Non-deterministic Polynomial time. Polynomial time means that the time to run algorithm is expressed by $O(n^k)$, where n is the size of inputs, and k some constant [15 pp. 840-841]. NP-hard problems are problems for which no polynomial time algorithm is known [15 page 719].

Due to being NP-hard, finding an optimal solution for the set cover will likely be intractable. Instead, a near-optimal solution with an approximation algorithm will be a better approach, e.g. in the set cover problem with large set sizes to keep execution time within a feasible limit. There exist many heuristic approximation algorithms for solving the set cover problem, such as: Greedy Algorithm (GA) [15 pp. 902-904], harmony search algorithm [16], ant colony optimization algorithm [17], various genetic algorithms [18][19][20], and more.

3.2.1 Greedy Algorithm

GA has been proven to run in polynomial time [15 pp. 902-904], which is why it was selected for this thesis. GA is an approximation algorithm that at each iteration selects the subset currently containing the largest number of uncovered elements. This is continued until all the elements are covered. The advantage of the GA is easy implementation and the fact that it can run in polynomial time while it can achieve results close to the optimal. [15 pp. 902-904]

It can be shown that the greedy algorithm does not always yield an optimal solution. For example, consider a set $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and subsets $A = \{0, 1, 2, 3, 4,5\}$, $B = \{0, 1, 5, 6, 7\}$, $C = \{2, 3, 4, 8, 9\}$, $D = \{6, 7, 8\}$, $E = \{9\}$. Initially, partial set is set to empty set, i.e. $P = \{\}$. Table 2, illustrates how the GA selects the subsets at each iteration.

Table 2. GA per iteration for the set U

Iteration #	Uncovered elements per subset per iteration	P	Subset with most uncovered
Initial	$A_0 = \{0, 1, 2, 3, 4, 5\}$ < $B_0 = \{0, 1, 5, 6, 7\}$ $C_0 = \{2, 3, 4, 8, 9\}$ $D_0 = \{6, 7, 8\}$ $E_0 = \{9\}$	$\{\}$	A_0
1	$A_1 = \{\}$ $B_1 = \{6, 7\}$ $C_1 = \{8, 9\}$ $D_1 = \{6, 7, 8\}$ < $E_1 = \{9\}$	A	D_1
2	$A_2 = \{\}$ $B_2 = \{\}$ $C_2 = \{9\}$ $D_2 = \{\}$ $E_2 = \{9\}$ <	$A \cup D$	C_2 or E_2
3	$A_3 = \{\}$ $B_3 = \{\}$ $C_3 = \{\}$ $D_3 = \{\}$ $E_3 = \{\}$	$A \cup D \cup E$	END

As seen from the Table 2, the GA arrived into a solution where the number of chosen subsets is three i.e. $P = A \cup D \cup E$. However, an optimal solution is two subsets, i.e. B and C.

3.2.2 Big-step Greedy Algorithm

Big-step Greedy Algorithm (BGA), works like the GA, but instead of selecting a single subset, BGA selects multiple subsets at each iteration. The number of selected subsets is determined by a step-size p. The BGA is suitable for applications where a marginal improvement is feasible even with increased run time. [21]

Consider the same set cover problem as in Section 3.2.1. Now, assume a BGA with step-size of two, i.e. p=2. Larger step-size results that there are more subset combinations to consider at each iteration. Table 3 illustrates how the BGA selects subsets at each iteration.

Table 3. BGA per iteration for the set U

Iteration #	Uncovered elements per subset pair per iteration	P	Subset pair with most uncovered
Initial	$A_0 \cup B_0$ = {0, 1, 2, 3, 4, 5, 6, 7} $A_0 \cup C_0$ = {0, 1, 2, 3, 4, 5, 8, 9} $A_0 \cup D_0$ = {0, 1, 2, 3, 4, 5, 6, 7, 8} $A_0 \cup E_0$ = {0, 1, 2, 3, 4, 5, 9} $B_0 \cup C_0$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} < $B_0 \cup D_0$ = {0, 1, 5, 6, 7, 8} $B_0 \cup E_0$ = {0, 1, 5, 6, 7, 9} $C_0 \cup D_0$ = {2, 3, 4, 6, 7, 8, 9} $C_0 \cup E_0$ = {2, 3, 4, 8, 9} $D_0 \cup E_0$ = {6, 7, 8, 9}	{}	$B_0 \cup C_0$
1	$A_0 \cup B_0$ = {} $A_0 \cup C_0$ = {} $A_0 \cup D_0$ = {} $A_0 \cup E_0$ = {} $B_0 \cup C_0$ = {} $B_0 \cup D_0$ = {} $B_0 \cup E_0$ = {} $C_0 \cup D_0$ = {} $C_0 \cup E_0$ = {} $D_0 \cup E_0$ = {}	$B \cup C$	END

In this scenario, the BGA was better than the GA and the achieved result was optimal. However, the BGA complexity quickly grows infeasibly large as the number of subsets grows large. In the example scenario, the number of subset combinations is already twice as large as the same scenario with the GA. The growth in complexity is exponential with respect to the numbers of subsets. Hence, the BGA is not practical to be used for set cover problems with a large number of subsets.

3.2.3 Weighed Greedy Algorithm

Weighed Greedy Algorithm (WGA) is like the GA. The difference between the WGA an GA is that, in the WGA, the elements of a set can have different weights or costs. The WGA is not applicable for a set cover problem where all the elements of the universe are found from the resulting subsets. Instead, WGA is suited for scenario where it is known that including all elements with a pre-determined number of subsets is not possible, and one wishes to prioritize some elements over the others.

For example, consider a practical scenario of forming a team of employees. Assume that there are 20 applicants with 30 different skills among them, and at most three applicants can be selected. Employer can value different skills differently, e.g. based on their relevance towards the work description, and assign different weights for the different skills. After a weight is

assigned for all the skills, each applicant has a selection weight, based on their skill set. Now, the employer may perform the WGA for the set of applicants, to find a set of three applicants that maximizes the number of different skills included to the team.

First, the WGA would choose the applicant with the largest weight, i.e., the one with most relevant skills. When choosing the second employee, the already included skills would be neglected, and the WGA would choose the next applicant with most relevant skills that are not yet included to the team. Similarly, the third employee is chosen based on the already included skills among the team. Resulting team will likely to be close to the optimal with respect to the relevant skills.

In the previous recruitment scenario, BGA with step size of three would result to the best possible team of three with respect to included skills. This is because the BGA would iteratively search the best set of three from the set of 20 people, which would ultimately be a brute force approach. The GA would suit in this scenario if the employer wanted to include all the skills with the minimum number of employees.

4 TEST RESOURCE OPTIMIZATION

Before the optimization of test resource utilization can be discussed, a test resource must be defined. Also, a way to denote the efficiency of a subset is required, before the optimality of the respective subset can be addressed.

4.1 Test resource definition

A test resource can have multiple meanings depending on the context. A test resource can be defined as: the number of the working-hours used by a tester, test equipment usage, or time taken in regression testing. In most cases, time is the most fundamental resource in testing. To maintain an appropriate level of abstractness, the concept of Test Resource Unit (TRU) is defined to be used as a metric of the resource usage of a test system. The TRU is an unitless unit that can be converted to any other resource measure by using ,e.g. conversion factors depending on the context to which the conversion is planned. TRU can be used to compare the resource usage of the different types of test systems on a same platform e.g. a group of test engineers and automated test environment.

4.2 Coverage metrics

In this thesis, the following metrics for coverage are defined: Combinatorial Coverage (CC), Independent Input Coverage (IIC), Output Coverage (OC), Non-Overlapping Input Coverage (NIC) and Combinatorial-NIC (C-NIC). Each of the coverage metrics are defined in a separate section.
To illustrate the different coverage metrics, a subset *S* and an example equation are defined. (4-1) is used as an example equation. (4-1) consists of three input parameters: a_1, a_2, a_3, where: $a_1, a_2, a_3 \in \{1,2,3\}$. A single input combination is of the form $\{a_1, a_2, a_3\}$. A subset is defined as a group of input combinations, e.g. $S = \{\{1, 1, 1\}, \{1, 2, 3\}, \{3, 2, 1\}, \{3, 3, 3\}\}$. The subset *S* consists of four input combinations of form $\{a_1, a_2, a_3\}$, and in first the combination of *S*, a_1=1, a_2=1, and a_3=1. Each input combination produces a single output value O.

$$O = (a_1 + a_2 + a_3)\mathrm{mod}(a_1 + a_2) \qquad (4-1)$$

Example subset $S = \{\{1, 1, 1\}, \{1, 2, 3\}, \{3, 2, 1\}, \{3, 3, 3\}\}$ is applied to the equation (4-1) to get the output values. The CC, IIC, NIC, C-NIC, and OC are calculated for this scenario in the subsequent sections.

4.2.1 Combinatorial Coverage

The CC is an input-specific coverage metric. CC compares the number of the input combinations contained in the included subset to all the possible input combinations. In this thesis, combinatorial coverage is referred as full combinational coverage i.e. N-way coverage, where *N* is the number of input parameters [22]. For a large set of input parameters, full CC is very hard, or not feasible, to achieve because the number of combinations grow exponentially for every added parameter. Although, the complexity also depends on the numbers of possible

values that the added parameters can obtain. If the parameters are mutually independent, the increment in the number of possible input combinations is equal to the multiplication of the number of the previous number of combinations and the possible values of the added parameter. In the example scenario, (4-1) consisted of three input parameters: a_1, a_2, a_3, each having three possible values. When the first parameter a_1 is added, the number of the possible combination is three: a_1 is either 1, 2, or 3. When the second, mutually independent parameter a_2, is added, the number of possible combinations is multiplied by the number of possible values a_2 can obtain, i.e. by three, resulting in nine possible combinations. Finally, a_3 is added, which further multiplies the number of possible combinations by three, resulting that the number of possible combinations for chosen input space is 27.

To calculate the CC for the example scenario, both the number of the included input combinations and the number of the possible input combinations must be known. As described before, the number of all possible input combinations, denoted as N_{cc}, for the example scenario was 27. The subset S was defined as S = {{1, 1, 1}, {1, 2, 3}, {3, 2, 1}, {3, 3, 3}}, i.e. the number of included combinations, denoted as $N_{\mathrm{inc,cc}}$, for the subset S is four. Respective CC is presented in the equation (4-2).

$$C_{\mathrm{cc}} = \frac{N_{\mathrm{inc,cc}}}{N_{\mathrm{cc}}} = \frac{4}{27} \qquad (4-2)$$

4.2.2 Individual input coverage

Just like the CC, IIC is an input-specific coverage metric. IIC could be seen as 1-way combinatorial coverage. Instead of considering all combinations between input parameters, IIC considers only the individual occurrences of parameters. In other words, the IIC is an average of the individual parameter coverages. The IIC coverage is defined in equation (4-3).

$$C_{\mathrm{iic}} = \frac{1}{N_{\mathrm{par}}} \sum_{i=1}^{N_{\mathrm{par}}} C_{\mathrm{iic},i} \qquad (4-3)$$

where, N_{par} is the number of parameters and $C_{\mathrm{iic},i}$ is the individual parameter coverage of i^{th} parameter. The individual parameter coverage, $C_{\mathrm{iic},i}$, is defined in equation (4-4).

$$C_{\mathrm{iic},i} = \frac{N_{\mathrm{inc,i}}}{N_{\mathrm{val,i}}} \qquad (4-4)$$

where, $N_{\mathrm{inc},i}$ is the number of different included values of i^{th} parameter, and $N_{\mathrm{val},i}$ is the number of all possible values i^{th} parameter can obtain.

In the equation (4-1), each input parameter could obtain three different values, i.e. the numbers of the possible values for input parameters a_1, a_2, a_3 are denoted as $N_{\mathrm{val},1}$, $N_{\mathrm{val}.2}$, $N_{\mathrm{val},3}$ = 3. The example subset S was defined as S = {{1, 1, 1}, {1, 2, 3}, {3, 2, 1}, {3, 3, 3}}. The number of different values included in S for parameters a_1, a_2, a_3, are denoted as: $N_{\mathrm{inc},1}$, $N_{\mathrm{inc},2}$, and $N_{\mathrm{inc},3}$, are $N_{\mathrm{inc},1}$=2, $N_{\mathrm{inc},2}$=3, and $N_{\mathrm{inc},3}$=2. The IIC for the subset S, denoted as $C_{\mathrm{iic},S}$, is defined in the equation (4-5).

$$C_{\text{iic},S} = \frac{1}{N_{\text{par}}} \sum_{i=1}^{N_{\text{par}}} \frac{N_{\text{inc},i}}{N_{\text{val},i}} \quad (4-5)$$

Now, N_{par}=3, as the combinations in the subset *S* consisted of three parameters. The numerical value of IIC for the subset *S* is calculated in the equations (4-6) and (4-7).

$$C_{\text{iic},S} = \frac{1}{3}\left(\frac{N_{\text{inc},1}}{N_{\text{val},1}} + \frac{N_{\text{inc},2}}{N_{\text{val},2}} + \frac{N_{\text{inc},3}}{N_{\text{val},3}}\right) \quad (4-6)$$

$$C_{\text{iic},S} = \frac{1}{3}\left(\frac{2}{3} + \frac{3}{3} + \frac{2}{3}\right) = \frac{7}{9} \quad (4-7)$$

4.2.3 Output coverage

Unlike CC and IIC, OC is an output-specific coverage metric. The OC coverage compares the numbers of different output values achieved, after inserting a subset into an equation, to the numbers of all the possible output values of the respective equation. The OC coverage is defined in the equation (4-8).

$$C_{\text{oc}} = \frac{N_{\text{inc,oc}}}{N_{\text{oc}}} \quad (4-8)$$

where, $N_{\text{inc,oc}}$ is the number of different output values achieved with inserted subset *S*, and N_{oc} is the number of all the possible output values of the equation (4-1). Table 4 presents the output values obtained after inserting *S* = {{1, 1, 1}, {1, 2, 3}, {3, 2, 1}, {3, 3, 3}} into the equation (4-1).

Table 4. Resulting output values after inserting subset *S* into equation (4-1)

Combination $= \{a_1, a_2, a_3\}$	Equation $O = (a_1 + a_2 + a_3)\text{mod}(a_1 + a_2)$	O
{1, 1, 1}	(1+1+1) mod (1+1) = 3mod2	1
{1, 2, 3}	(1+2+3) mod (1+2) = 6mod3	0
{3, 2, 1}	(3+2+1) mod (3+2) = 6mod5	1
{3, 3, 3}	(3+3+3) mod (3+3) = 9mod6	3

To calculate the OC, knowledge about the different possible output values of the equation (4-1) is required. The number of all possible output values is not immediately trivial due to the same parameter affecting both dividend and divisor. To obtain the number of possible output values, the Equation (4-1) was implemented with python after which all the different input combinations were inserted into it and all the resulting different output values were counted. Based on the code execution, the equation (4-1) had four different possible values: 0, 1, 2, and 3. The code used to obtain the result is presented in the Figure 6.

```
1
2  different_output_values = []
3
4  for a1 in [1, 2, 3]:
5      for a2 in [1, 2, 3]:
6          for a3 in [1, 2, 3]:
7              output = (a1 + a2 + a3)%(a1 + a2)
8              if output not in different_output_values:
9                  different_output_values.append(output)
10
11 print(sorted(different_output_values))
12 print(len(different_output_values))
```

Figure 6. Code that finds all different output values for equation (4-1) .

The number of achieved output values, after inserting S into the equation (4-1) was $N_{\text{inc,oc}} = 3$, and the number of all possible output values $N_{\text{oc}} = 4$. The numeric value of OC for subset S is calculated in the equation (4-9).

$$C_{\text{oc}} = \frac{N_{\text{inc,oc}}}{N_{\text{oc}}} = \frac{3}{4} \qquad (4-9)$$

4.2.4 Non-overlapping input coverage

NIC coverage is an equation-specific, or formula specific, coverage metric. The NIC is different than CC, IIC, and OC coverages in a sense that it ignores both input and output values. Instead, the NIC considers the states of the equation terms. An equation state means a situation in which all the terms of an equation have obtained a combination of values. A term state means a situation in which a specific section of an equation, denoted as term, obtains a specific value.

To define NIC, term coverages must be defined. A term coverage refers to the ratio of the numbers of different included term states of k^{th} term, $N_{\text{term,state},k}$, to the numbers of all possible states, denoted as $N_{\text{term,state,all},k}$, of k^{th} term. The term coverage for k^{th} term, $C_{\text{term},k}$, is defined in the equation (4-10).

$$C_{\text{term},k} = \frac{N_{\text{term,state},k}}{N_{\text{term,state,all},k}} \qquad (4-10)$$

Note that $N_{\text{term,state},k}$ and $N_{\text{term,state,all},k}$ in the equation (4-10) denote the number of numerical values that an equation term can achieve and not the all the combinations among the term parameters. NIC equals an average of term coverages of an equation. NIC is defined in the equation (4-11).

$$C_{\text{nic}} = \frac{1}{N_{\text{term}}} \sum_{k=1}^{N_{\text{term}}} C_{\text{term},k} \qquad (4-11)$$

where, N_{term} is the number of the equation terms, and $C_{\text{term},k}$ is the term coverage of the k^{th} term.

The equation (4-1) can be divided into two terms as follows: $T_1 = (a_1 + a_2 + a_3)$ and $T_2 = (a_1 + a_2)$. To calculate the NIC, the number of obtained T_1 and T_2 states, denoted by N_{T1} and N_{T2}, and all the possible term states, denoted by $N_{T1,\text{all}}$ and $N_{T2,\text{all}}$, must be known. Table 5 presents the numbers of obtained term states after the subset *S* was inserted into the equation (4-1) .

Table 5. Obtained term states when subset *S* was inserted to equation (4-1)

Combination ={a_1, a_2, a_3}	$T_1 = (a_1 + a_2 + a_3)$	$T_2 = (a_1 + a_2)$
{1, 1, 1}	3	2
{1, 2, 3}	6	3
{3, 2, 1}	6	5
{3, 3, 3}	9	6

From the Table 5, $N_{T1} = 3$ and $N_{T2} = 4$. All possible values for terms T_1 and T_2 equals the range of integer numbers between their minimum and maximum values as all the contained parameters were integers in range {1, 2, 3} and the terms consisted of only summation. Therefore, T_1 can obtain all the integer values in the range {3 . . . 9}, i.e. seven different values. Respectively, T_2 can obtain all integer values in the range {2 . . . 6}, i.e. five different values. Hence, $N_{T1,\text{all}} = 7$ and $N_{T2,\text{all}} = 5$. The term coverages, denoted as C_{T1} and C_{T2}, are calculated in the equations (4-12) and (4-13).

$$C_{T1} = \frac{N_{T1}}{N_{T1,\text{all}}} = \frac{3}{7} \qquad (4-12)$$

$$C_{T2} = \frac{N_{T2}}{N_{T2,\text{all}}} = \frac{4}{5} \qquad (4-13)$$

The NIC is obtained by inserting the obtained C_{T1} and C_{T2} into the equation (4-11). Numerical value of the NIC for subset S is calculated in the equation (4-14).

$$C_{nic} = \frac{1}{2}(C_{T1} + C_{T2}) = \frac{1}{2}\left(\frac{3}{7} + \frac{4}{5}\right) = \frac{43}{70} \qquad (4-14)$$

4.2.5 Combinatorial – Non-overlapping input coverage

Like NIC, C-NIC is an equation-specific coverage metric. C-NIC is related to NIC, in similar way as IIC is related to CC. C-NIC is fundamentally an *N*-way combination coverage for the equation states, where the *N* is the number of considered equation sections. Instead of considering the individual term coverages of an equation, C-NIC considers the combinations of the term states. C-NIC is defined in the equation (4-15):

$$C_{\text{cnic}} = \frac{N_{\text{states,eq}}}{N_{\text{states,eq,all}}} \qquad (4-15)$$

C-NIC is a measure of equation state coverage. It is possible that a single equation state can be achieved with various input combinations, which has the potential to greatly reduce the numbers of required combinations to achieve all the states of an equation. For example, the equation (4-

1) achieves same state with combinations {1, 2, 1} and {2, 1, 1}, meaning that all the equation states can be achieved without going through all the input combinations.

Table 6, illustrates all the possible combinations of the parameters a_1, a_2, a_3 and the resulting equation states after inserting the subset *S* into the equation (4-1) . In the Table 6, the equation states included in subset *S* are denoted by a green background, and overlapping equation states, in C-NIC sense, with a grey background.

Table 6. C-NIC combinations of Equation (4-1)

a_1	a_2	a_3	$(a_1 + a_2 + a_3)$	$(a_1 + a_2)$	Equation state #
1	1	1	3	2	1
1	1	2	4	2	2
1	1	3	5	2	3
1	2	1	4	3	4
1	2	2	5	3	5
1	2	3	6	3	6
1	3	1	5	4	7
1	3	2	6	4	8
1	3	3	7	4	9
2	1	1	4	3	4
2	1	2	5	3	5
2	1	3	6	3	6
2	2	1	5	4	7
2	2	2	6	4	8
2	2	3	7	4	9
2	3	1	6	5	10
2	3	2	7	5	11
2	3	3	8	5	12
3	1	1	5	4	7
3	1	2	6	4	8
3	1	3	7	4	9
3	2	1	6	5	10
3	2	2	7	5	11
3	2	3	8	5	12
3	3	1	7	6	13
3	3	2	8	6	14
3	3	3	9	6	15

From the Table 6, the equation (4-1) can obtain 15 possible states in C-NIC sense, four of which was achieved with the subset *S*. It follows that, $N_{\text{states,eq}}$=4, $N_{\text{states,eq,all}} = 15$. C-NIC is obtained after inserting $N_{\text{states,eq}}$ and $N_{\text{states,eq,all}}$ into the equation (4-15). The numerical value of C-NIC for the subset *S* is calculated in the equation (4-16).

$$C_{\text{cnic}} = \frac{N_{\text{states,eq}}}{N_{\text{states,eq,all}}} = \frac{4}{15} \qquad (4-16)$$

4.2.6 Overall Coverage

In this thesis, overall coverage is used as the ultimate coverage metric. Overall coverage is an average of IIC, C-NIC, and OC coverages. Overall coverage is defined in the equation (4-17). IIC, C-NIC, OC were chosen because they, in conjunction, cover all the inputs, outputs, and states of a system, thus, giving a good overall estimate about the coverage of the chosen system. OC considers: "how the outside world sees the system". C-NIC considers: "what happens inside the system". IIC considers: "how the system sees the world". Therefore, the system is inspected from three different perspectives, giving a good overall comprehension of the coverage, or in other words, the ratio of "what has happened" to "everything that is possible to happen".

$$C_{\text{overall}} = \frac{C_{\text{oc}} + C_{\text{cnic}} + C_{\text{iic}}}{3} \qquad (4-17)$$

(4-17) is a discrete increasing function. All the components in the equation (4-17) can be increased with a single resource, i.e. the components are not independent. All the coverage components can achieve values from 0 to 1. The value ranges of the components are discrete, and the density of values depends on the number of possible input values, C-NIC combinations, and output values. Ideally, if enough resources are available, all the components achieve full coverage, i.e. all input values, C-NIC combinations, and outputs are considered in testing, and C_{overall}=1. Also, overall coverage will achieve positive non-zero value for every positive non-zero number of resources, as only a single resource is enough to include new input, C-NIC combination, and output as nothing is initially tested.

In the equation (4-17), the coverage components were non-weighed. This results that if one of the coverage metrics is much easier to achieve than others, it will dominate the overall coverage when a small number of resources are available. Different weight factors for the coverage metrics could be also be used. There are two approaches to select the weights. First way is to assign weights based on the importance of the coverage metrics so that the largest weight is assigned to the most important coverage. Second way is to assign weights based on the hardness to achieve full coverage for the respective metric. Problem with latter approach is that the definition of overall coverage would be different for different equations.

If weights were to be added, their sum should be in the divisor of the equation (4-17), to normalize the coverage values. Adding weights would remove the domination of a single coverage metric in case of a small number of available resources (or increase it if the weights were chosen based on importance), but simultaneously it would result that the hardest coverage would dominate cases where there are a large number of resources available. Unless full coverage for all the coverage metrics are equally difficult to achieve, there will be bias

In this thesis, weights are not used as each coverage metric is considered as equally important and to maintain comparability to other equations. The overall coverage of the example subset can be calculated by inserting C_{oc}, C_{cnic} and C_{iic}, obtained from the previous sections to the equation (4-17). The numerical value of overall coverage for subset *S* is calculated in the equation (4-18).

$$C_{\text{overall}} = \frac{\frac{3}{4} + \frac{4}{15} + \frac{7}{9}}{3} = \frac{323}{540} \qquad (4-18)$$

4.3 Subset efficiency metrics

An efficiency metric, used to determine the subset efficiency throughout this thesis, is defined in this section. An efficiency is a measure of how effectively the resources were utilized. More precisely, the efficiency describes how well the resources were transferred into coverage. An efficiency that utilizes a coverage as metric, is denoted as Coverage Based Efficiency (CBE). CBEs can be separated based on the used coverage metrics, defined in the section 4.2. CBE metrics include: CC-efficiency, IIC-efficiency, OC-efficiency, NIC-efficiency, C-NIC-efficiency, and overall efficiency. General definition of all the CBEs is defined in the equation (4-19).

$$\varepsilon_{\mathrm{cbe}} = \frac{C}{R} \tag{4-19}$$

where,
C = Achieved coverage of selected coverage metric
R = Resource usage expressed in TRUs

To illustrate CBE efficiency metrics, all CBEs for the subset S are calculated. Coverage results of the subset S for the equation (4-1), were presented in the previous section. In this case, a single TRU shall be defined as a single input combination applied to an equation. Hence, the resource usage of the subset S is four. Numerical values of the respective CC-, IIC-, OC-, NIC-, C-NIC-, and overall-efficiencies are calculated in the equations (4-20) – (4-25).

$$\varepsilon_{\mathrm{cc}} = \frac{C_{\mathrm{cc}}}{R} = \frac{0.148}{4} = 0.037 \tag{4-20}$$

$$\varepsilon_{\mathrm{iic}} = \frac{C_{\mathrm{iic}}}{R} = \frac{0.778}{4} = 0.1945 \tag{4-21}$$

$$\varepsilon_{\mathrm{oc}} = \frac{C_{\mathrm{oc}}}{R} = \frac{0.75}{4} = 0.1875 \tag{4-22}$$

$$\varepsilon_{\mathrm{nic}} = \frac{C_{\mathrm{nic}}}{R} = \frac{0.614}{4} = 0.1535 \tag{4-23}$$

$$\varepsilon_{\mathrm{cnic}} = \frac{C_{\mathrm{cnic}}}{R} = \frac{0.266}{4} = 0.0666 \tag{4-24}$$

$$\varepsilon_{\mathrm{overall}} = \frac{C_{\mathrm{overall}}}{R} = \frac{0.598}{4} = 0.1495 \tag{4-25}$$

4.3.1 Relative and optimal efficiency

The efficiency results alone, do not provide much information about the efficiency of subset S, other than the coverage per resource information, as there is no reference. Hence, a relative efficiency, ε_{r}, is introduced. To define the relative efficiency, a reference point must also be determined. Optimal overall efficiency, $\varepsilon_{\mathrm{opt}}$, is selected as the reference point. The optimal overall efficiency is a quantity that describes the best possible achievable efficiency for a given system. Optimal efficiency is used as a reference when determining the relative efficiency with respect to the achieved $\varepsilon_{\mathrm{cbe}}$. Relative efficiency is defined as the ratio of the achieved efficiency $\varepsilon_{\mathrm{cbe}}$ to the optimal overall efficiency $\varepsilon_{\mathrm{opt}}$, as expressed in the equation (4-26).

$$\varepsilon_{\mathrm{r}} = \begin{cases} \dfrac{\varepsilon_{\mathrm{cbe}}}{\varepsilon_{\mathrm{opt}}}, & R \leq R_{\min} \\ \dfrac{\varepsilon_{\mathrm{cbe}}}{\varepsilon_{\mathrm{opt}}} \dfrac{R_{\min}}{R}, & R > R_{\min} \end{cases} \qquad (4-26)$$

Optimal overall efficiency $\varepsilon_{\mathrm{opt}}$, depends on the subset size R. To define the $\varepsilon_{\mathrm{opt}}$, two extreme points of efficiency must be defined: optimal discrete efficiency and optimal full coverage efficiency. The optimal discrete efficiency, denoted as $\varepsilon_{\mathrm{opt,single}}$, is defined as the maximum possible coverage increase per a single resource. The optimal discrete efficiency is defined in equation (4-27).

$$\varepsilon_{\mathrm{opt,single}} = \frac{C_{\mathrm{max,single}}}{R_{\min}} = C_{\mathrm{max,single}} \qquad (4-27)$$

where, $C_{\mathrm{max,single}}$ is the maximum possible increase in a coverage with a single TRU, and $R_{\min}$ = 1, because the discrete efficiency considered an efficiency of a single TRU at a time. The optimal full coverage efficiency, denoted as $\varepsilon_{\mathrm{opt,full}}$, is defined as a minimum TRUs required to achieve a full coverage for the selected coverage metric. The optimal full coverage efficiency is defined in equation (4-28).

$$\varepsilon_{\mathrm{opt,full}} = \frac{C_{\max}}{R_{\min}} = \frac{1}{R_{\min}} \qquad (4-28)$$

where, $C_{\max}$ is one because maximum coverage is considered and $R_{\min}$ is the minimum number of resources required to achieve a full coverage.

In addition, a threshold θ must be defined. The threshold θ is the maximum subset size that can achieve the optimal discrete efficiency $\varepsilon_{\mathrm{opt,single}}$. The exact value of θ can be determined by considering how many combinations can be included to the subset, while each added combination increases all the coverage metrics by the maximum possible amount. The threshold θ is constrained by the coverage metric that can be achieved with the least number of resources. For example, if a system had only two possible outputs, the respective θ for the system will be at most two, because after two combinations it is not possible to include new output values, following that the next added combination cannot increase the coverage more than the previous two did. For systems for which determining the threshold θ is not manually feasible, e.g. due to complex dependencies between the input parameters, θ can be approximated as in the equation (4-29).

$$\theta \approx \frac{1}{\varepsilon_{\mathrm{opt,single,max}}} \qquad (4-29)$$

where, $\varepsilon_{\mathrm{opt,single,max}}$ is the optimal discrete efficiency of the coverage metric that can be achieved with the least number of combinations. Optimal efficiency approximation $\varepsilon_{\mathrm{opt}}$ for the overall coverage of R-size subset is defined in the equation (4-30).

$$\varepsilon_{\text{opt}}(R) \approx \begin{cases} \varepsilon_{\text{opt,single}}, & R \leq \theta \\ \frac{1}{R}\left(C_{\text{inc,oc}}N_{\text{inc,oc}} + C_{\text{inc,cnic}}N_{\text{inc,cnic}} + \sum_{i=1}^{N_{\text{par}}} C_{\text{inc},i}N_{\text{inc},i}\right), & \theta < R \leq R_{\min} \\ \varepsilon_{\text{opt,full}}\frac{R_{\min}}{R}, & R > R_{\min} \end{cases} \quad (4-30)$$

where,
$C_{\text{inc,oc}}$ = Increase in overall coverage for each new output,
$C_{\text{inc,cnic}}$ = Increase in overall coverage for each new C-NIC combination,
$C_{\text{inc},i}$ = Increase in overall coverage for each new value of i^{th} parameter,
$N_{\text{inc,oc}}$ = Number of different output values included in the subset,
$N_{\text{inc,cnic}}$ = Number of different C-NIC combinations included in the subset,
$N_{\text{inc},i}$ = Number of different values of i^{th} input parameter included in subset, and
$N_{\text{par}} =$ Number of system variables

(4-30) is an approximation function because the calculated value of the $R_{\min}$ cannot always be proved to be the exact value as the calculation of the $R_{\min}$ is fundamentally a set cover problem, which is a NP-hard problem as discussed in section 3.2. Similarly, the calculated values of $N_{\text{inc,oc}}$, $N_{\text{inc,oc}}$, and $N_{\text{inc},i}$ cannot be prooved to be the exact values for the same reason. The second section, $\frac{1}{R}\left(C_{\text{inc,oc}}N_{\text{inc,oc}} + C_{\text{inc,cnic}}N_{\text{inc,cnic}} + \sum_{i=1}^{N_{\text{par}}} C_{\text{inc},i}N_{\text{inc},i}\right)$, of the equation (4-30) could be used as an approximation formula for all values of *R*. The definition of the equation (4-30) was divided into three sections, to better illustrate the values θ and $\varepsilon_{\text{opt,full}}$. The division was done also because the exact value of θ is possible to determine, which means that in the range $R \leq \theta$, $\varepsilon_{\text{opt,single}}$ is the exact solution for the optimal efficiency. For $R > R_{\min}$, the term: $C_{\text{inc,oc}}N_{\text{inc,oc}} + C_{\text{inc,cnic}}N_{\text{inc,cnic}} + \sum_{i=1}^{N_{\text{par}}} C_{\text{inc},i}N_{\text{inc},i} = C_{\max} = 1$. This means that the second section of the equation (4-30) reduces to $\frac{1}{R}$, which is equal to the third section if the definition of overall coverage, from the equation (4-28), is inserted into the third section, i.e. $\frac{1}{R_{\min}}\frac{R_{\min}}{R} = \frac{1}{R}$. Fundamentally, the third section is a simplified version of the notation of the second section of the equation (4-30).

The coverage increases for each new output, C-NIC combination, and input value, $C_{\text{inc,oc}}$, $C_{\text{inc,cnic}}$, and $C_{\text{inc},i}$ are defined in the equations: (4-31), (4-32), and (4-33).

$$C_{\text{inc,oc}} = \frac{1}{N_{\text{oc}}}\frac{1}{3} \quad (4-31)$$

$$C_{\text{inc,cnic}} = \frac{1}{N_{\text{cnic}}}\frac{1}{3} \quad (4-32)$$

$$C_{\text{inc,par},i} = \frac{1}{N_{\text{val},i}}\frac{1}{N_{\text{par}}}\frac{1}{3} \quad (4-33)$$

where, $N_{\text{val},i}$ is the number of the different possible values of the i^{th} input parameter. Coverage increase for each output or C-NIC value is inversely proportional to the numbers of the different possible values of the respective metric. The term $\frac{1}{3}$ comes from the fact that the overall

coverage, in this thesis, is an unweighted average of the three coverage metrics. Coverage increase for a single new parameter value, depends on the number of the possible values of the parameter, and the total number of input parameters.

Approximations of the number of included output values, $N_{\text{inc,oc}}$, C-NIC combinations, $N_{\text{inc,cnic}}$, and input parameter values, $N_{\text{inc},i}$, are defined in the equations (4-34), (4-35), and (4-36), respectively.

$$N_{\text{inc,oc}} \approx \min\left(\frac{N_{\text{oc}}}{R_{\text{min,oc}}}R, N_{\text{oc}}\right) \quad (4-34)$$

$$N_{\text{inc,cnic}} \approx \min\left(\frac{N_{\text{cnic}}}{R_{\text{min,cnic}}}R, N_{\text{cnic}}\right) \quad (4-35)$$

$$N_{\text{inc},i} \approx \min\left(\frac{N_{\text{val},i}}{R_{\text{min,par},i}}R, N_{\text{val},i}\right) \quad (4-36)$$

where, $R_{\text{min,oc}}$ is the minimum number of resources required for full OC, $R_{\text{min,cnic}}$ the minimum number of resources required for full C-NIC, and $R_{\text{min,par},i}$ the minimum number of resources required for full IIC, for the i^{th} parameter. The estimations presented in the equations (4-34), (4-35), and (4-36), consider the average numbers of new entries per added resource, based on the ratio of all possible values to the minimum numbers of resources required to include all the values.

4.3.2 Numeric example

To illustrate the calculation of the efficiency metrics, the optimal-, and the relative efficiencies are calculated for the equation (4-1), with respect to overall coverage. First, the system parameters, $\varepsilon_{\text{opt,single}}$, $\varepsilon_{\text{opt,full}}$, R_{min}, and θ are determined. Second, coverage increases $C_{\text{inc,oc}}$, $C_{\text{inc,cnic}}$, and $C_{\text{inc,par},i}$ are calculated with respect to overall coverage. Third, the approximation formulas for $N_{\text{inc,oc}}$, $N_{\text{inc,cnic}}$, and $N_{\text{inc,par},i}$ are simplified for the equation (4-1). Fourth, the obtained values are used to calculate an approximation of optimal subset efficiency as a function of subset size R. Fifth, GA with different selection thresholds is used to find optimal and sub-optimal subsets for the equation (4-1). The obtained results are compared to the optimal approximation. Finally, the relative efficiency for subset *S* is calculated. Later in this section, efficiency and overall coverage efficiency is used interchangeably. Also, coverage and overall coverage are used interchangeably.

The optimal discrete efficiency, $\varepsilon_{\text{opt,single}}$, is equal to the maximum possible coverage increase for a single combination. For equation (4-1), the optimal discrete efficiency is achieved when a new combination includes: a new output, a new C-NIC, and a new entry for all the input parameters. The numerical value for the $\varepsilon_{\text{opt,single}}$ is calculated in the equations (4-37) and (4-38).

$$\varepsilon_{\text{opt,single}} = C_{\text{max,single}} = \frac{1}{3}\left(\frac{1}{N_{\text{oc}}} + \frac{1}{N_{\text{cnic}}} + \frac{1}{N_{\text{par}}}\sum_{i=1}^{N_{\text{par}}}\frac{1}{N_{\text{val},i}}\right) \quad (4-37)$$

$$\varepsilon_{\text{opt,single}} = \frac{1}{3}\left(\frac{1}{4} + \frac{1}{15} + \frac{1}{3}\left(\frac{1}{3} + \frac{1}{3} + \frac{1}{3}\right)\right) = \frac{13}{60} \quad (4-38)$$

Minimum resources $R_{\min}$ was the number of resources required to achieve full coverage. In the section 4.2.5, it was mentioned that the number of possible C-NIC combinations, for the equation (4-1), was 15. Given that all the possible output and input values can be included with 15 combinations, minimum resources can be declared to be 15. One possibility to select a full coverage subset with 15 resources is presented in the Table 7. In the Table 7, full IIC section is denoted with blue, full C-NIC with yellow, and full OC with green.

Table 7. One way to select a full coverage subset using 15 resources

a_1	a_2	a_3	$(a_1 + a_2 + a_3)$	$(a_1 + a_2)$	Equation state #	Output
1	1	1	3	2	1	1
2	2	2	6	4	2	2
3	3	3	9	6	3	3
1	1	2	4	2	4	0
1	1	3	5	2	5	1
1	2	1	4	3	6	1
1	2	2	5	3	7	2
1	2	3	6	3	8	0
1	3	1	5	4	9	1
1	3	3	7	4	10	3
2	3	1	6	5	11	1
2	3	2	7	5	12	2
2	3	3	8	5	13	3
3	3	1	7	6	14	1
3	3	2	8	6	15	2

The optimal full coverage efficiency, $\varepsilon_{\text{opt,full}}$, was a quantity describing the average coverage increase per resource for a full coverage subset. The numerical value of optimal full coverage efficiency is presented in the equation (4-39).

$$\varepsilon_{\text{opt,full}} = \frac{1}{R_{\min}} = \frac{1}{15} \qquad (4-39)$$

The system parameter θ, described the maximum subset size that can achieve overall efficiency equal to the maximum discrete efficiency. For the equation (4-1), θ can be determined from Table 7 by noticing that the first three combinations achieve new entry for all coverage metrics, meaning that fourth included parameter would achieve discrete optimal efficiency strictly less than $\varepsilon_{\text{opt,single}}$. θ can also be approximated by using the equation (4-29). In this scenario, the quantity $\varepsilon_{\text{opt,single,max}}$ is the discrete optimal efficiency of IIC coverage i.e. $\frac{1}{3}$. Approximation of θ is presented in the equation (4-40). In this scenario, the approximated value of θ is equal to the exact value, extracted from Table 7.

$$\theta \approx \frac{1}{\varepsilon_{\text{opt,single,iic}}} = \frac{1}{\frac{1}{3}} = 3 \qquad (4-40)$$

Next, the coverage increases for a new output, a new C-NIC combination, and each new input parameter value, are calculated. From the Table 7, $N_{\mathrm{oc}} = 4$, $N_{\mathrm{cnic}} = 15$, $N_{\mathrm{val},a1} = 3$, $N_{\mathrm{val},a2} = 3$, $N_{\mathrm{val},a3} = 3$, and $N_{\mathrm{par}} = 3$. All the input parameters had an equal number of input values, i.e. $N_{\mathrm{val},i} = 3$, for all the input parameters. The respective coverage increases are obtained by introducing these values to the equations: (4-31), (4-32), and (4-33). Coverage increase calculations are presented in the equations: (4-41), (4-42), (4-43).

$$C_{\mathrm{inc,oc}} = \frac{1}{N_{\mathrm{oc}}}\frac{1}{3} = \frac{1}{4}\frac{1}{3} = \frac{1}{12} \tag{4-41}$$

$$C_{\mathrm{inc,cnic}} = \frac{1}{N_{\mathrm{cnic}}}\frac{1}{3} = \frac{1}{15}\frac{1}{3} = \frac{1}{45} \tag{4-42}$$

$$C_{\mathrm{inc},a1} = C_{\mathrm{inc},a2} = C_{\mathrm{inc},a3} = \frac{1}{N_{\mathrm{val},i}}\frac{1}{N_{\mathrm{par}}}\frac{1}{3} = \frac{1}{3}\frac{1}{3}\frac{1}{3} = \frac{1}{27} \tag{4-43}$$

Approximation equations: (4-34), (4-35), and (4-36), can be simplified in the case of equation (4-1), because a single combination can only achieve at most one new entry for each coverage metric. The simplified versions of the approximation equations are presented in the equations (4-44), (4-45), and (4-46).

$$N_{\mathrm{inc,oc}} \approx \min(R, N_{\mathrm{oc}}) \tag{4-44}$$

$$N_{\mathrm{inc,cnic}} \approx \min(R, N_{\mathrm{cnic}}) \tag{4-45}$$

$$N_{\mathrm{inc},i} \approx \min\left(R, N_{\mathrm{val},i}\right) \tag{4-46}$$

Simplified approximations assume that, for an optimal subset, each added combination provides a new entry for the respective coverage metric, until the number of added resources is equal to the number of the possible different values of the elements for the respective metric.

By using the calculated values for coverage increments, and the numbers of included element approximations, the optimal efficiency for equation (4-1) was calculated as a function of subset size R. In addition, optimal and sub-optimal subsets were searched with GA, by assigning selection thresholds 5, 4, 3, 2, 1, and 0. Selection threshold is the minimum number of total new elements that must be found from a combination for the combination to be selected, i.e. the sum of new output, C-NIC, and input values. An optimal subset can be found if the initial selection threshold is set to five, because a single combination can at most include one new output, C-NIC and three new input values, one for each input parameter. The selection threshold is decreased by one, after each combination is iterated with previous threshold, until all the missing elements are found. Sub-optimal subsets can be searched by assigning initial threshold smaller than five. Figure 7 presents the calculated optimal efficiency approximation and the actual subset efficiencies obtained by GA.

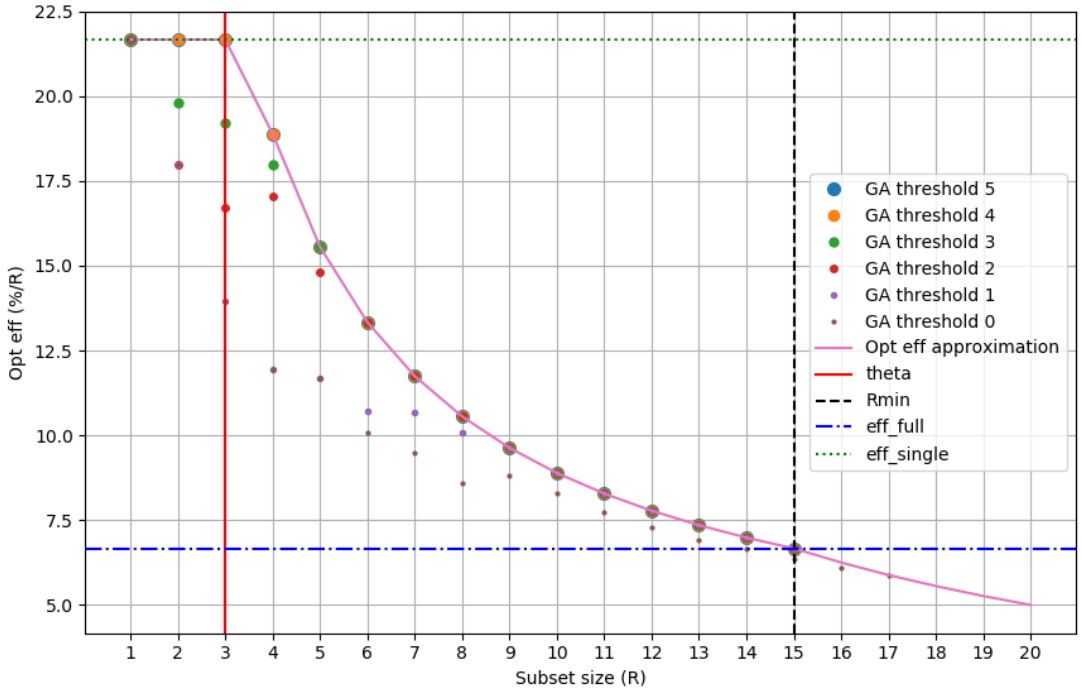

Figure 7. Optimal efficiency approximation and achieved values for GA

From Figure 7, results for the GA with threshold of five are equal to the approximated optimal efficiency. However, if the selection threshold was smaller than five, the efficiencies of the selected subsets are no longer optimal for all subset sizes R. Although, the subsets obtained with GA, where initial threshold was smaller than five, still converged towards the optimal efficiency as the subset size increased.

Convergence occurs because, the number of missing combinations quickly decreases, as the included combinations include missing combinations at least equal to the GA threshold. However, in the presented scenario, there was also an element of luck involved, as if even a single combination without a new C-NIC combination would be included before $R_{\min}$ resources, the efficiency would not equal the optimal before $R_{\min}$ resources. Although in the Figure 7, GA with threshold 4 happened to achieve optimal efficiency for all subset sizes R. For example, with threshold of four, one output, C-NIC, or input value can be left out, when a combination containing all new values existed in the CS. With initial thresholds larger than two, GA still appears to select a new C-NIC at each iteration because optimal efficiency is achieved before $R_{\min}$ resources are included. For initial threshold smaller than two, GA selects such subset in which there are overlapping C-NIC combinations, resulting into sub-optimal efficiency also after $R_{\min}$ resources.

Note that GA, is not guaranteed to find optimal solution even when initial threshold is five because it is possible to select first combination so that it is no longer possible to select two more combinations so that both contain new output, C-NIC, and three new input values. Such scenario occurs e.g. if two first input combinations were: $\{1, 1, 3\}$, and $\{2, 3, 2\}$. This behaviour is illustrated in Figure 8, where GA with different initial thresholds was ran 1000 times, and the combinations of the CS were shuffled between runs. Plotted result is an aggregation of the worst results for each of the subset sizes R.

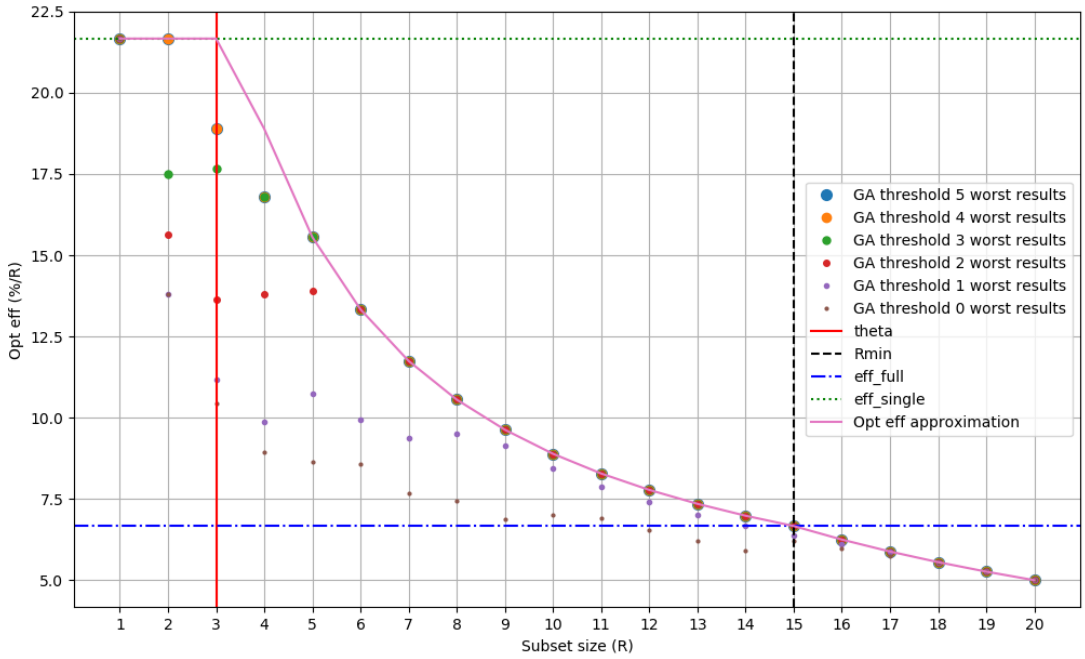

Figure 8. Worst efficiency results for GA with various initial selection thresholds.

After the optimal efficiency is calculated, the relative efficiency of subset S can be determined. From the equation (4-25), it is known that the efficiency of subset S was, $\varepsilon_{\mathrm{cbe},S} = 0.1495$, it is also known that the subset size R of S was four. The Figure 9, presents the comparison of the efficiency of the subset S and the optimal efficiency. As it can be seen from Figure 9, the efficiency of subset S is not optimal, as the optimal efficiency for subset of size four is $\varepsilon_{\mathrm{opt}}(4) = 0.1889$. Relative efficiency for subset S is presented in equation (4-47).

$$\varepsilon_{\mathrm{r}} = \frac{\varepsilon_{\mathrm{cbe},S}}{\varepsilon_{\mathrm{opt}}(4)} = \frac{0.1495}{0.1889} = 0.7914 \qquad (4-47)$$

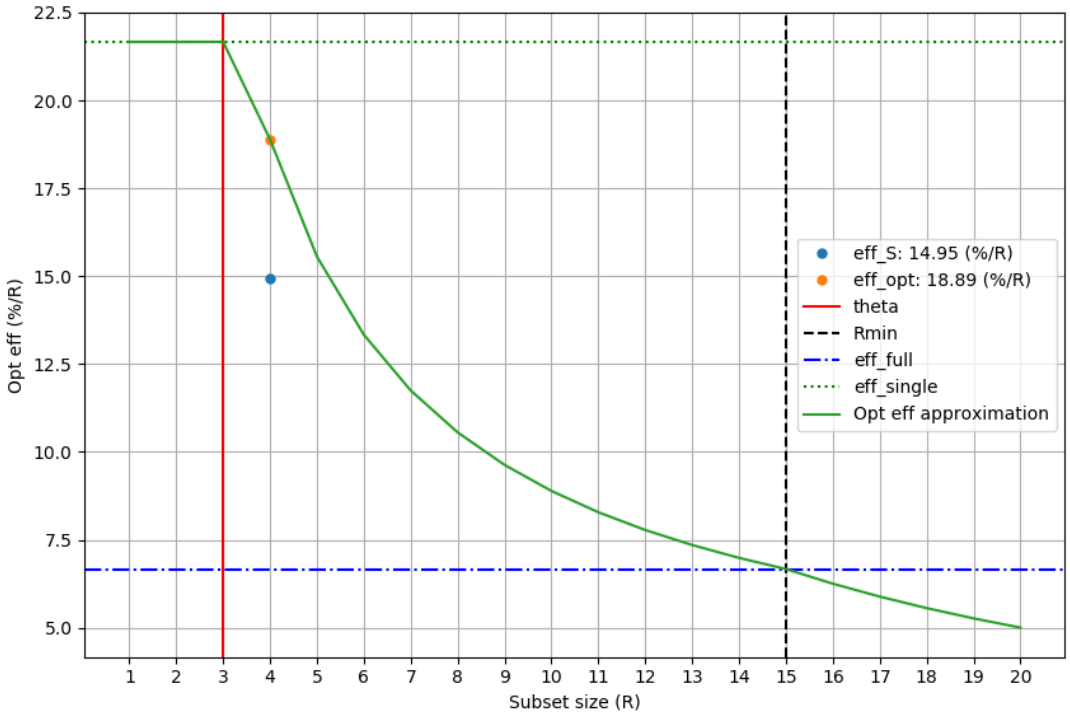

Figure 9. Efficiency of subset S compared to optimal efficiency.

From (4-47) result, and from Figure 9, it can be concluded that the efficiency of subset S is quite far from the optimal. A subset of size four, that would achieve the optimal efficiency would be e.g. Q = {{1, 1, 1}, {2, 2, 2}, {3, 3, 3}, {1, 1, 2}}, which can be verified from the Table 7.

4.4 Subset performance

Efficiency, or coverage alone, is a poor estimation of the performance of a selected subset. Coverage provides only the information about the absolute performance of the chosen subset but does not indicate whether the resources were spent wisely, i.e. could the same coverage have been achieved with less resources. Conversely, the efficiency only describes the relative performance of a subset, leaving the absolute performance unknown.
Coverage-efficiency relation could be described with an analogy about a mountain climber who wishes to reach the top of a mountain by climbing the shortest possible route. If the climber only knew his coverage, he would know exactly the distance to the top, but he would have no idea if the path he chose was the optimal. Conversely, if the climber only knew his efficiency, he would know the lengths of all routes to the top, including the one he chose, but he would have no knowledge about the current distance to the top. If the climber knew both his coverage and efficiency he would have a good comprehension about his overall performance as he would know both the length of the chosen route and the distance to the top.

Instead of considering coverage or efficiency separately, combination of both efficiency and coverage gives a better estimation of the overall performance for a chosen subset. Subset performance described via CBE metrics is denoted with P_{cbe}. As mentioned, both efficiency and coverage should be used to evaluate the subset performance. The subset performance is estimated with overall efficiency, coverage and a complexity of the CS of the evaluated

equation, denoted by a hardness factor *h*. The hardness factor is used to emphasize the coverage when size of CS is large. Hardness factor is used to scale the distance to the full coverage subset for different sized CS. Hardness factor is also used to value coverage increase for a complex system more than for same increase for a simple system. In this thesis, the hardness factor h is defined as in equation (4-48).

$$h = \log_{10}(\log_{10} N_{\mathrm{C}}) \qquad (4-48)$$

where, N_{C} is the number of possible ways to select $R_{\min}$ from all possible combinations. Different form of the definition of h is presented in equation (4-49).

$$h = \log_{10}(\log_{10}(\mathrm{nCr}(N_{\mathrm{cc}}, R_{\min}))) \qquad (4-49)$$

Where the operation $\mathrm{nCr}(N_{\mathrm{cc}}, R_{\min})$ refers to the number of ways to select a non-ordered set of $R_{\min}$ from a set of N_{cc}. N_{cc} for the example scenario was calculated to be 27, in the section 4.2.1. The numeric value for the hardness factor for the example scenario is calculated in the equation (4-50).

$$h = \log_{10}(\log_{10}(\mathrm{nCr}(27,15))) = 0.85975 \qquad (4-50)$$

Next, two performance metrics are introduced: absolute performance P_{cbe} and relative performance $P_{\mathrm{cbe,r}}$. The absolute performance is a metric for the level of test resource utilization, whereas the relative performance is a metric describing the distance to an optimal subset for testing, i.e. full coverage with full efficiency. Absolute performance P_{cbe} is defined in the equation (4-51).

$$P_{\mathrm{cbe}}(R) = \varepsilon_{\mathrm{cbe}} C^{h} \qquad (4-51)$$

From the equation (4-8), $C = 0.5981$, from the equation (4-25), $\varepsilon_{\mathrm{cbe}} = 0.1495$, and from the equation (4-50), $h = 0.85975$. The numerical value for absolute performance for the subset S is calculated in the equation (4-52).

$$P_{\mathrm{cbe},S} = \varepsilon_{\mathrm{cbe},S} C^{h} = (0.1495)(0.5981)^{0.85975} = 0.0961 \qquad (4-52)$$

Result from (4-52) alone, does not provide much information. Yet, it can be used to compare the level of resource utilization between the subsets of different sizes and for different systems. For example, the absolute performance of an optimal full coverage subset, denoted as *Q,* can be compared to that of subset *S*, to compare the levels of resource utilization. The subset *Q* can be formed by selecting all combinations from the Table 7. The numerical value of absolute performance for subset *Q* is calculated in the equation (4-53).

$$P_{\mathrm{cbe},Q} = \varepsilon_{\mathrm{opt,full}} {C_{\mathrm{overall}}}^{h} = \frac{1}{15} 1^{0.85975} = \frac{1}{15} = 0.0666 \qquad (4.-53)$$

The result of the equation (4-53) indicates that the absolute performance of subset *S* is better than that of subset *Q,* i.e. the resource utilization of *S* is better than of the subset *Q*, although the coverage of subset *Q* was larger. Further, the absolute performance of an optimal subset of size four can be calculated to compare the absolute performances of equally-sized subsets. One

option for an optimal subset of size four is B = {{1, 1, 1}, {2, 2, 2}, {3, 3, 3}, {1, 1, 2}}. Subset B, achieves full IIC, full OC, and includes four different C-NIC combinations, which can be verified from the Table 7. The respective overall coverage is calculated in the equation (4-54), and the numerical value of overall efficiency in the equation (4-55).

$$C_{\text{overall},B} = \frac{1}{3}\left(1 + 1 + \frac{4}{15}\right) = 0.7555 \qquad (4-54)$$

$$\varepsilon_{\text{cbe},B} = \frac{C_{\text{overall},B}}{4} = 0.1889 \qquad (4-55)$$

By inserting the results of (4-50), (4-54) and (4-55), into the equation (4-51), the value of absolute performance for the subset B can be calculated, as presented in the equation (4-56).

$$P_{\text{cbe},B} = \varepsilon_{\text{cbe},B} {C_{\text{overall},B}}^{h} = (0.1889)(0.7555)^{0.85975} = 0.1484 \qquad (4-56)$$

Based on the result of the equation (4-56), the resource utilization of the subset B is better than that of the subset S, or an optimal full coverage subset. The result indicates that even the optimal full coverage subset might not be the best option with respect to test resource utilization. Instead, there might exist a subset, smaller than $R_{\min}$, that optimizes the test resource utilization. For further reference, the absolute performances of a subset that contains all the combinations, denoted as subset E, is calculated in equation (4-57).

$$P_{\text{cbe},E} = \varepsilon_{\text{cbe},E} {C_{\text{overall}}}^{h} = \frac{1}{15}\frac{15}{27}(1)^{0.85975} = 0.0370 \qquad (4-57)$$

Not surprisingly, the absolute performance of subset E, that tests every combination, is worse than the absolute performance of subset Q, that achieved full coverage with less resources. Absolute performance results are summarized in the Table 8.

Table 8. Absolute performance results for subsets: S, B, O, and E, for equation (4-1)

Subset	R	ε_{cbe}	C_{overall}	P_{cbe}
S	4	0.1495	0.5981	0.0961
B	4	0.1889	0.7555	0.1484
Q	15	0.0666	1.0000	0.0666
E	27	0.0370	1.0000	0.0370

Next, relative performance is considered. Relative performance, $P_{cbe,r}$, is defined in equation (4-58). Relative performance describes the ratio of achieved performance to the optimal full coverage subset. Relative performance can be used to evaluate the distance to the optimal testing subset i.e. to a subset with full coverage with maximum efficiency.

$$P_{\text{cbe,r}}(R) = \varepsilon_{\text{r}} C^{h} \qquad (4-58)$$

The numerical values of the relative performances of subsets S, B, Q, and E are calculated, to illustrate the quantity $P_{\text{cbe,r}}$. First, the approximated optimal efficiency of the corresponding subset sizes must be calculated with equation (4-30). The relative efficiencies can be calculated by inserting the obtained optimal approximations and achieved efficiencies to the equation (4-26). The Table 9 presents the calculated relative efficiencies, achieved efficiencies, and

respective relative efficiencies if the subsets *S, B, Q,* and *E.* The Table 10 presents the numerical values of the relative performances that were obtained by inserting the relative efficiencies from the Table 9, and the overall coverage values from the Table 8, into the equation (4-58).

Table 9. Relative efficiency results for subsets: *S, B, Q,* and *E,* for equation (4-1)

Subset	R	ε_{opt}	ε_{cbe}	ε_{r}
S	$4 \leq 15$	0.1889	0.1495	0.7914
B	$4 \leq 15$	0.1889	0.1889	1.0000
Q	$15 \leq 15$	0.0666	0.0666	1.0000
E	$27 > 15$	0.0370	0.0370	0.5555

Table 10. Relative performance results for subsets: *S, B, Q,* and *E,* for equation (4-1)

Subset	ε_{r}	C_{overall}	$P_{\text{cbe,r}}$
S	0.7914	0.5981	0.5087
B	1.000	0.7555	0.7858
Q	1.000	1.000	1.000
E	0.5555	1.000	0.5555

To further illustrate the system performance metric P_{cbe}, performance results for an optimal subset, approximated with equation (4-30), is presented in the Figure 10. Blue line illustrates the relative subset performance, orange line is the absolute subset performance, green line is the discrete efficiency per added resource, red line is the coverage, and purple line is the optimal efficiency approximation. In the Figure 10, the relative efficiency is one for all subset sizes R, i.e. subset efficiency is equal to the optimal efficiency at each R. Note that values in the Figure 10 are normalized to keep focus at the shapes and maximum points.

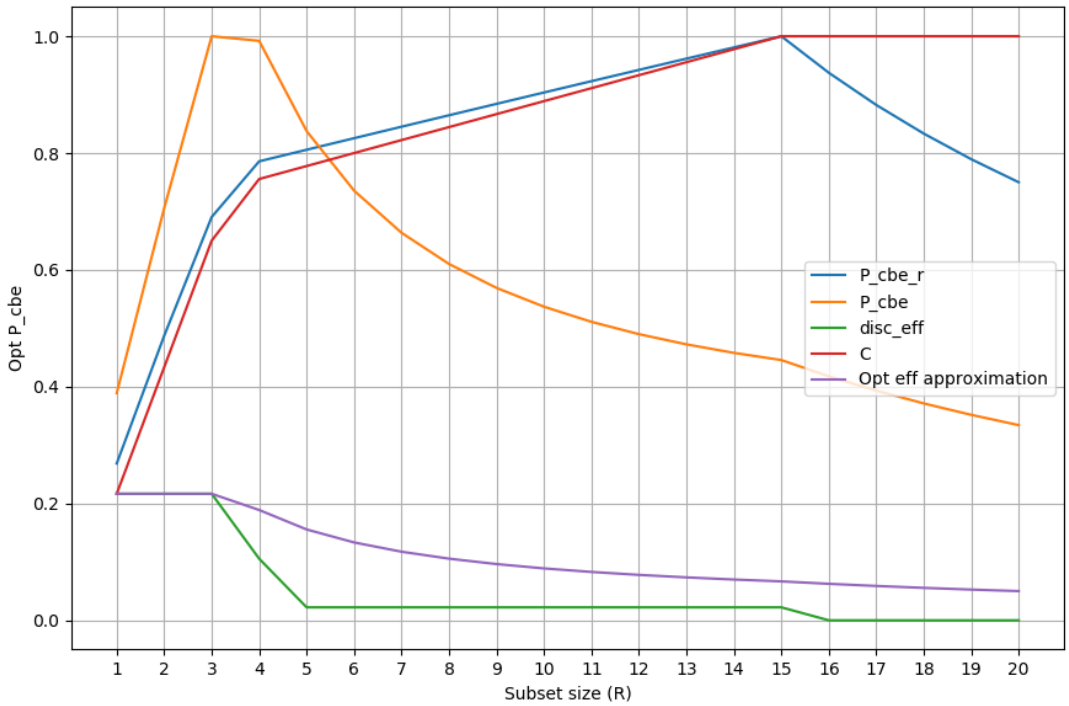

Figure 10. P_{cbe}, $P_{\text{cbe,r}}$ $\varepsilon_{\text{opt\_discrete}}$, ε_{opt} and coverage of an optimal subset for equation (4-1), as a function of the subset size R.

Three observations can be made from the Figure 10. First, a subset of size 15 achieves the maximum relative performance, i.e. full coverage at optimal relative efficiency. After $R_{\min}$ resources, the relative performance starts to degrade because full coverage is achieved using more resources than necessary, although relative efficiency is optimal. Using more than $R_{\min}$ resources is not feasible as e.g. subset size R=4 achieves better performance than e.g. R=20. Also, the optimal efficiency decreases slower than the coverage increases, causing the steady increase in performance when resources are added, until $R_{\min}$ resources.

Second, the relation of discrete efficiency, optimal efficiency and coverage can be observed. When $R < \theta$, coverage increments are very large for each added resource, and both discrete- and optimal efficiencies are constant. After θ threshold is exceeded, discrete efficiency begins to decrease quickly, which decreases the coverage increment for each added resource. The decrease of optimal efficiency is slower as it measures the average increase in coverage for each subset size R. After $R_{\min}$ resources, the discrete efficiency drops to zero as the coverage can no longer be increased.

Third, the absolute performance is maximized at three resources, which is understandable as by optimally selecting three combinations resource utilization is maximized as each added combination includes new values with respect to every considered coverage metric. The point, at which the absolute performance is maximized, is still far from the optimal full coverage subset.

4.5 C-NIC decomposition

C-NIC decomposition divides an equation into sections that achieve values from a finite set based on the given inputs. The formed sections are denoted as NIC terms which, in conjunction with the other NIC terms, form a single C-NIC combination. The benefit of the C-NIC decomposition is that the complexity of an equation can potentially be reduced by excluding the repeated values of any equation term.

Ultimately, a single NIC term is a change of variables operation. NIC terms should be formed for any part of equation that can obtain the same value with different input combinations. Equation with the modulo operator is a good example where forming NIC terms is feasible. General rules for C-NIC composition are listed below:

1) Equation state information must be preserved, i.e. the formation of NIC terms must not cause equation states to become hidden.
2) Complexity must reduce, i.e. obtained C-NIC decomposition must have less combinations than the number of the possible input combinations of original equation.
3) NIC terms do not partially overlap, i.e. the original equation must be possible to express with the formed NIC terms.

For example, consider the equation (4-59), where $a_i = \{1, 2, 3\}\ \forall\ i \in \{1, 2, 3\}$. An example of invalid NIC term formation is presented in equations (4-60) and (4-61).

$$O = (a_1 + a_2)\mathrm{mod}(a_2)\mathrm{mod}(a_1 a_3) \quad (4-59)$$

$$NIC_1 = (a_1 + a_2)\mathrm{mod}(a_2) \quad (4-60)$$

$$NIC_2 = (a_2)\mathrm{mod}(a_1 a_3) \quad (4-61)$$

Because the information of the possible states of the terms $a_1 + a_2$ in (4-60) and $a_1 a_3$ in (4-61) would become hidden, condition 1 is violated. Term a_2 overlaps between the two NIC terms resulting that the original equation cannot be expressed with chosen NIC terms, thus violating condition 3. To meet the conditions 1 and 3, the NIC terms should instead be selected as illustrated in the equations (4-62), (4-63), and (4-64).

$$NIC_1 = a_1 + a_2 \quad (4-62)$$
$$NIC_2 = (NIC_1) \bmod (a_2) \quad (4-63)$$
$$NIC_3 = a_1 a_3 \quad (4-64)$$

Now, the conditions 1 and 3 are met because the term state information of $a_1 + a_2$ and $a_1 a_3$ is preserved and the original equation is possible to express via chosen NIC terms, i.e. $O = (NIC_2) \bmod (NIC_3)$.

Finally, it must be verified that the condition 2 is also met. The original equation had 27 possible input combinations. If the number of possible C-NIC combinations is less than 27, the C-NIC decomposition is valid. Table 11, presents all the possible NIC term and C-NIC values for all input combinations, for the equation (4-59).

Table 11. (4-59) C-NIC combinations

a_1	a_2	a_3	NIC_1	NIC_2	NIC_3	C-NIC #
1	1	1	2	0	1	1
1	1	2	2	0	2	2
1	1	3	2	0	3	3
1	2	1	3	1	1	4
1	2	2	3	1	2	5
1	2	3	3	1	3	6
1	3	1	4	1	1	7
1	3	2	4	1	2	8
1	3	3	4	1	3	9
2	1	1	3	0	2	10
2	1	2	3	0	4	11
2	1	3	3	0	6	12
2	2	1	4	0	2	13
2	2	2	4	0	4	14
2	2	3	4	0	6	15
2	3	1	5	2	2	16
2	3	2	5	2	4	17
2	3	3	5	2	6	18
3	1	1	4	0	3	19
3	1	2	4	0	6	15
3	1	3	4	0	9	20
3	2	1	5	1	3	21
3	2	2	5	1	6	22
3	2	3	5	1	9	23
3	3	1	6	0	3	24
3	3	2	6	0	6	25
3	3	3	6	0	9	26

As it can be seen from Table 11, complexity decreases by one combination i.e. condition 2 is not violated. Therefore, all conditions are met and C-NIC decomposition can be declared valid.

Note that the decrease in the absolute complexity increases with respect to the combination space size e.g. with $a_i = \{1, 2, \ldots, 8, 9\}\ \forall\ i \in \{1, 2, 3\}$, there are 729 possible input combinations and 684 C-NIC combinations, i.e. decrease of 45 combinations. For $a_i = \{1, 2, \ldots, 28, 29\}\ \forall\ i \in \{1, 2, 3\}$, 24 389 input combinations and 23 264 C-NIC combinations, i.e. decrease of 1125 combinations.

5 APPLICATION TO 3GPP EQUATION

In this chapter, a 3GPP equation is selected and the search of an optimal subset for testing the equation is attempted by searching a subset that achieves maximum absolute performance. Chosen equations is, PDSCH hopping narrowband calculation for Bandwidth limited / Coverage Enhanced (BL/CE) UE. The equation was chosen due to being sufficiently complex for a proof-of-concept trial, and the frequent usage of the modulo operator, which potentially allows for simplifications to take place.

5.1 Equation Introduction

NB Hopping calculation for PDSCH is defined in [12 pp. 84-85]. Version used in this thesis is V13.10.0, i.e. baseline is June 2018. The PDSCH hopping narrowband calculation is presented in equation (5-1) [12 pp. 84-85].

$$n_{\mathrm{NB}}^{(i)} = \left(n_{\mathrm{NB}}^{(i_0)} + \left(\left\lfloor \frac{i + i_{\Delta}}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} - j_0 \right\rfloor \mathrm{mod} N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}} \right) \cdot f_{\mathrm{NB,hop}}^{\mathrm{DL}} \right) \mathrm{mod} N_{\mathrm{NB}}^{\mathrm{DL}} \qquad (5-1)$$

$$j_0 = \left\lfloor (i_0 + i_{\Delta}) / N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right\rfloor,$$

$$i_0 \leq i \leq i_0 + N_{\mathrm{abs}}^{\mathrm{PDSCH}} - 1,$$

$$i_{\Delta} = \begin{cases} 0, & \text{for frame structure type 1} \\ N_{\mathrm{NB}}^{\mathrm{ch,DL}} - 2, & \text{for frame structure type 2} \end{cases},$$

where,

i_0	= absolute subframe number of the first downlink subframe intended for PDSCH,
$n_{\mathrm{NB}}^{(i_0)}$	= narrowband of first absolute subframe intended for PDSCH,
$N_{\mathrm{NB}}^{\mathrm{ch,DL}}$	= Number of consecutive absolute subframes over which MPDCCH or PDSCH stays at the same narrowband before hopping to another narrowband, expressed as number of absolute subframes,
$N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$	= Number of narrowbands over which MPDCCH or PDSCH frequency hops,
$f_{\mathrm{NB,hop}}^{\mathrm{DL}}$	= Narrowband offset between one narrowband and the next narrowband an MPDCCH or PDSCH hops to, expressed as number of downlink narrowbands,
$N_{\mathrm{NB}}^{\mathrm{DL}}$	= Number of downlink narrowbands, and
$N_{\mathrm{abs}}^{\mathrm{PDSCH}}$	= Total number of absolute subframes over which PDSCH with repetition spans.

The equation (5-1), generates a cyclic hopping pattern that repeats every $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$ subframes in time domain. To cover all the output combinations, considering the first consecutive i_0 values equal to the max value of $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$ is enough to include all the possible output pattern combinations, as following i_0 values would generate identical patterns on the resource grid, only on different location in time domain. Luckily, because the starting

narrowband is fixed with parameter $n_{\mathrm{NB}}^{(i_0)}$, the required number of consecutive i_0 values reduce to the maximum value of hopping interval, $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, as the frequency domain pattern repeats every $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ consecutive i_0 values, although the patterns itself can differ over 32 subframes. Without fixed starting narrowband, the starting narrowband could change if $i_0 > N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ condition is true, resulting that $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$ consecutive i_0 would be needed. This is illustrated in Figure 11, in which different patterns are indicated with red and green boxes. In red scenario, $i_0 < N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, resulting that starting narrowband would remain the same. In green scenario $i_0 > N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ condition is true, meaning that starting narrowband would change without fixed starting narrowband. In practice, the starting narrowband in green scenario is same as in the red scenario.

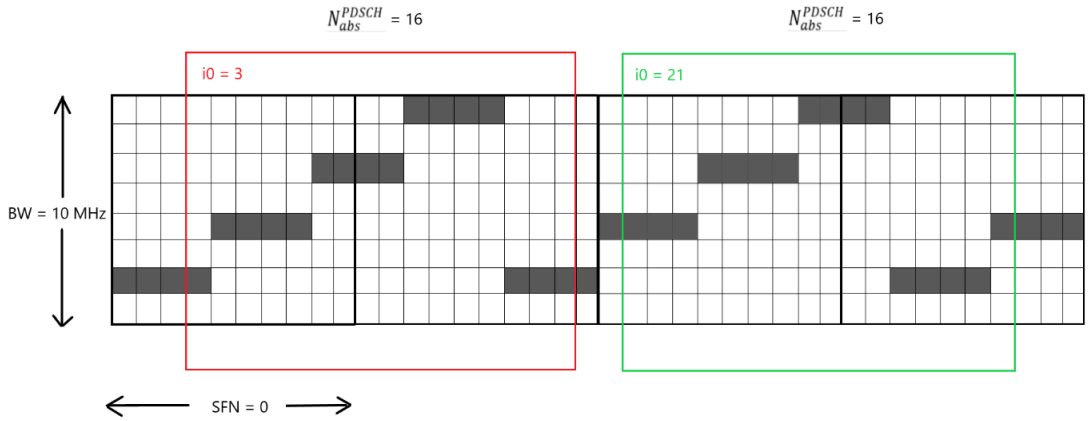

Figure 11. Hopping pattern without fixed starting narrowband.

The number of different output pattern combinations for a single input combination with respect to different i_0 values is determined in the equation (5-2).

$$N_{\mathrm{comb}} = \min\left(N_{\mathrm{NB}}^{\mathrm{ch,DL}}, N_{\mathrm{abs}}^{\mathrm{PDSCH}}\right) \qquad (5-2)$$

Table 12 presents the ranges of possible values, for the equation (5-1) parameters, gathered from 3GPP 36.331. For ranges presented in the Table 12, FDD and Coverage Enhanced Mode A (CE-ModeA) is assumed. The CE-ModeA means a configuration where the utilization of time diversity via repeated transmissions is possible in the network. In CE-ModeA, the network can send up to 32 repetitions of PDSCH message.

Table 12. Possible values for equation (5-1) parameters

Symbol	Higher-layer parameter name	Range
i_0	-	0…10239
$n_{\mathrm{NB}}^{(i_0)}$	-	1…$N_{\mathrm{NB}}^{\mathrm{DL}}$
$N_{\mathrm{NB}}^{\mathrm{ch,DL}}$	Interval-DLHoppingConfigCommonModeA-r13 [23 page 266]	1, 2, 4, 8
$N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$	mpdcch-pdsch-HoppingNB-r13 [23 page 266]	2, 4
$f_{\mathrm{NB,hop}}^{\mathrm{DL}}$	mpdcch-pdsch-HoppingOffset-13 [23 page 266]	1…$N_{\mathrm{NB}}^{\mathrm{DL}}$
$N_{\mathrm{NB}}^{\mathrm{DL}}$	dl-Bandwidth [23 page 230]	(1*), 2, 4, 8, 12, 16

$N_{\text{abs}}^{\text{PDSCH}}$	-	$\geq N_{\text{rep}}^{\text{PDSCH}}$

* No NB hopping if only one narrowband is available.

5.2 C-NIC Decomposition

The definition of the possible values for the parameter $N_{\text{abs}}^{\text{PDSCH}}$ is problematic as it considers also non-BL/CE DL subframes in which PDSCH transmission is postponed [30]. For simplicity, it is assumed that $N_{\text{abs}}^{\text{PDSCH}} = N_{\text{rep}}^{\text{PDSCH}}$, i.e. no non-BL/CE DL subframes are present, which results that possible values for $N_{\text{abs}}^{\text{PDSCH}}$, in case of FDD and CEModeA, are: 1, 2, 4, 8, 16, and 32. [24].

(5-1) has 198 819 840 possible input combinations [Appendix 1], which means that the testing of all the input combinations is not feasible. Yet, the number of equation states might be a much smaller number. To be able to form the C-NIC terms, C-NIC decomposition is performed to the equation (5-1). Initially, equation (5-1) can be divided into two terms from modulo operation. First term, denoted by T_1, is presented in equation (5-3).

$$T_1 = \left\lfloor \frac{i + i_\Delta}{N_{\text{NB}}^{\text{ch,DL}}} - j_0 \right\rfloor \text{mod} N_{\text{NB,hop}}^{\text{ch,DL}} \qquad (5-3)$$

T_1 can achieve values: 0, 1, 2, and 3. For FDD, i_Δ is always 0, which is why it can be ignored in this case. To not lose any NIC term information, T_1 is further be divided into two sections, denoted by $T_{1.1}$ and $T_{1.2}$, as presented in equations (5-4) and (5-5).

$$T_{1.1} = \left\lfloor \frac{i}{N_{\text{NB}}^{\text{ch,DL}}} - j_0 \right\rfloor \qquad (5-4)$$

$$T_{1.2} = N_{\text{NB,hop}}^{\text{ch,DL}} \qquad (5-5)$$

Second term, denoted by T_2, is presented in equation (5-6). T_2 is further divided into two sections, denoted by $T_{2.1}$ and $T_{2.2}$, as presented in equations (5-7) and (5-8).

$$T_2 = \left(n_{\text{NB}}^{(i_0)} + (T_1) \cdot f_{\text{NB,hop}}^{\text{DL}}\right) \text{mod} N_{\text{NB}}^{\text{DL}} \qquad (5-6)$$

$$T_{2.1} = n_{\text{NB}}^{(i_0)} + (T_1) \cdot f_{\text{NB,hop}}^{\text{DL}} \qquad (5-7)$$

$$T_{2.2} = N_{\text{NB}}^{\text{DL}} \qquad (5-8)$$

Full C-NIC coverage is achieved if all possible combinations between the states of $T_{1.1}$, $T_{1.2}$, $T_{2.1}$, and $T_{2.2}$ are gone through. Table 13, presents the numbers of different possible values for the terms $T_{1.1}$, $T_{1.2}$, $T_{2.1}$, and $T_{2.2}$.

Table 13. The number of combinations for the terms $T_{1.1}$, $T_{1.2}$, $T_{2.1}$, and $T_{2.2}$

Section	Combinations
$T_{1.1}$	32
$T_{1.2}$	2
$T_{2.1}$	64
$T_{2.2}$	5

Based on the Table 13, there are 20480 different C-NIC combinations, if the terms were mutually independent, which is significantly less compared to the number of CC combinations. If the term dependencies are inspected closer, it is noticed that $f_{\mathrm{NB,hop}}^{\mathrm{DL}}$, $n_{\mathrm{NB}}^{(i_0)}$, and $N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$ depend on the number of available narrowbands, $N_{\mathrm{NB}}^{\mathrm{DL}}$, meaning that the number of C-NIC combinations reduce further as some of the combinations, between the $T_{1.1}$, $T_{1.2}$, $T_{2.1}$, and $T_{2.2}$, are not possible.

Figure 12 illustrates an algorithm that finds a subset achieving full C-NIC coverage for equation (5-1). The algorithm was run for the equation (5-1), with respective parameter values from the Table 12. The resulting subset had 4944 different C-NIC combinations [Appendix 2].

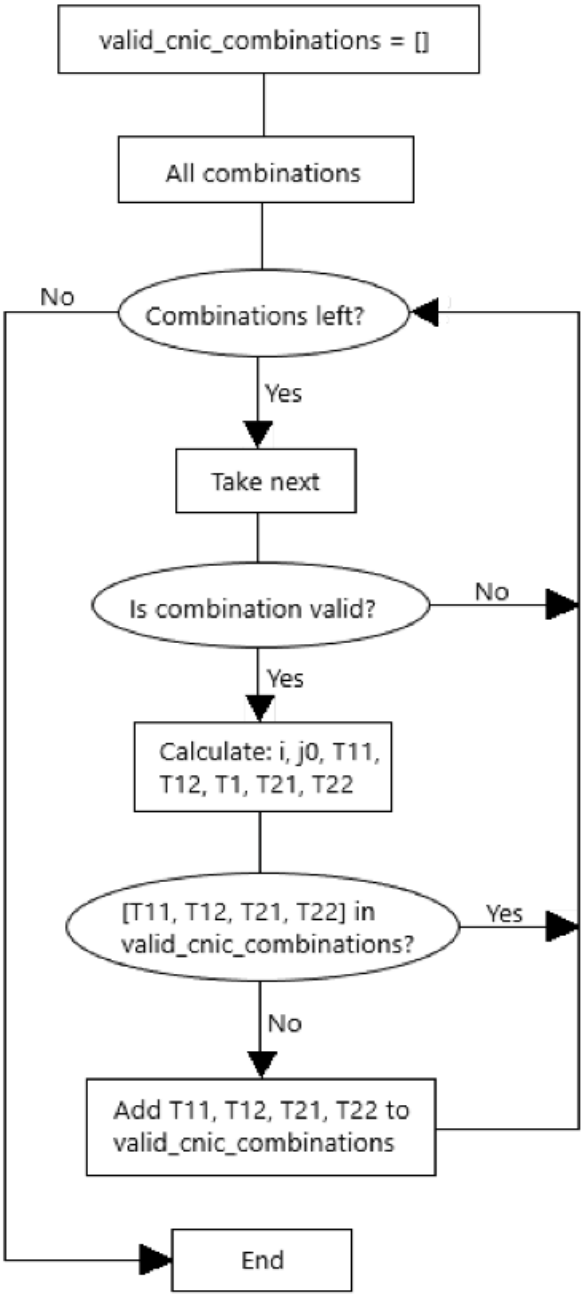

Figure 12. Algorithm for finding full C-NIC coverage achieving subset.

5.3 Output Coverage

In the OC sense, all the output patterns must be considered. A single input combination, generates an output pattern of length $N_{\text{abs}}^{\text{PDSCH}}$. As stated before, eight consecutive i_0 values are enough to find all output combinations. This information is helpful as it reduces the search space significantly. Figure 13 presents an algorithm that finds all the possible output patterns for the equation (5-1).

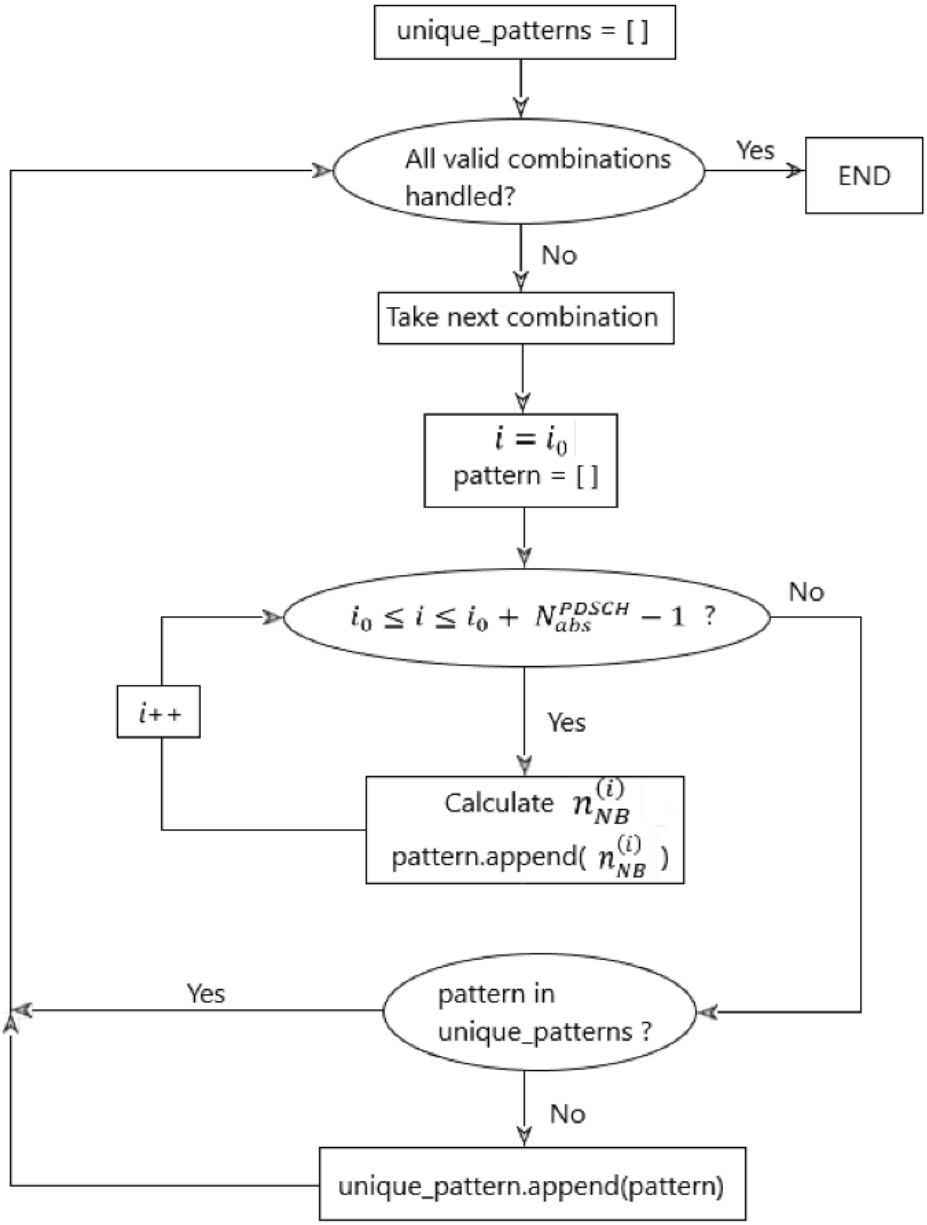

Figure 13. Algorithm that finds all possible output patterns for equation (5-1).

When the algorithm, illustrated in Figure 13, was implemented and executed with python, there were 24 336 different output patterns [Appendix 3]. This result was achieved with the first eight values of i_0. Running the algorithm for all the 10240 values of i_0, does not change the result. Only the execution of the algorithm would take 1280 times longer and the result would be 1280 identical pattern sets repeating every eight subframes, from which, only the first set would be stored.

5.4 Individual Input Coverage

In IIC point of view, the range for i_0 can be limited to not include multiple identical output pattern sets. The range is limited according to the parameter having second most possible values, i.e. 16. For example, the parameter for the starting narrowband, $n_{\mathrm{NB}}^{(i_0)}$, had 16 different

possible values. 16 consecutive i_0 values is sufficient in C-NIC and OC sense as full coverages for them can be achieved with only 8 consecutive i_0 values. Algorithm that finds full IIC subset, with limited i_0 range, with a minimum amount of combinations is presented in Appendix 4.

When the algorithm was executed, the resulting subset consisted of 16 combinations, which achieves full IIC coverage when range for i_0 was from 0 to 15. The full IIC subset is presented in Table 14. With restricted range of i_0, the CS of the equation (5-1) reduced to 370 176 combinations, which is used as the CS in this thesis from now on [Appendix 5].

Table 14. Full IIC achieving subset for i_0 in range 0…15

Combination #	$N_{\mathrm{NB}}^{\mathrm{DL}}$	$n_{\mathrm{NB}}^{i_0}$	$N_{\mathrm{NB}}^{\mathrm{ch,DL}}$	$N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$	$f_{\mathrm{NB,hop}}^{\mathrm{DL}}$	i_0	$N_{\mathrm{abs}}^{\mathrm{PDSCH}}$
1	2	2	1	2	2	0	1
2	4	4	2	4	4	2	2
3	8	8	4	2	8	1	4
4	12	12	8	4	12	3	8
5	16	16	1	2	16	4	16
6	2	1	2	2	1	5	32
7	4	3	4	2	3	6	1
8	8	7	8	4	7	7	2
9	12	11	1	2	11	8	4
10	16	15	2	4	15	9	8
11	8	6	4	2	6	10	16
12	12	10	8	4	10	11	32
13	16	14	1	2	14	12	1
14	8	5	4	4	5	13	2
15	12	9	2	2	9	14	4
16	16	13	8	4	13	15	8

5.5 Overall Coverage

Overall coverage was selected as the main coverage metric in this thesis because overall coverage contains an aggregated information about the C-NIC, OC, and IIC metrics. To find an optimal subset for testing, considering all the coverage metrics in parallel is desired. Full C-NIC subset required 4 944 combinations, full OC required 24 336 combinations, and full IIC required 10 240 combinations, if all the values of i_0 were considered. To find an optimal subset in overall coverage sense, a subset that achieves full C-NIC, OC, IIC, must be found. Found subset can be used as initial subset from which combinations are selected with pre-determined resource limit.

Parallel optimization is illustrated in the Figure 14. In the Figure 14, the outer cube represents an optimal full coverage subset. The edge of the outer cube is equal to R_{min}. The yellow cube represents a subset size of R. Each added resource increases the diagonal length of the yellow cube by $\sqrt{3}$ .The blue hexagon, that is partially inside the yellow cube, represents all the elements that can be found from the CS. The dark green hexagon represents the elements that are possible to include into the subset of size R. The part of the blue hexagon, that is outside the yellow cube, represents the elements that cannot be included in the subset with R resources. The small, orange, hexagon inside the dark green hexagon represents the included elements of subset size R, in which the elements have been selected inefficiently. Graphically, the overall

coverage is the volume ratio of the orange hexagon to the blue hexagon. Absolute efficiency is a ratio of the orange hexagon to the yellow hexagon. Relative efficiency is the volume ratio of the orange hexagon to the dark green hexagon. One way to graphically describe optimization is trying to maximize the volume ratio of orange hexagon and yellow cube.

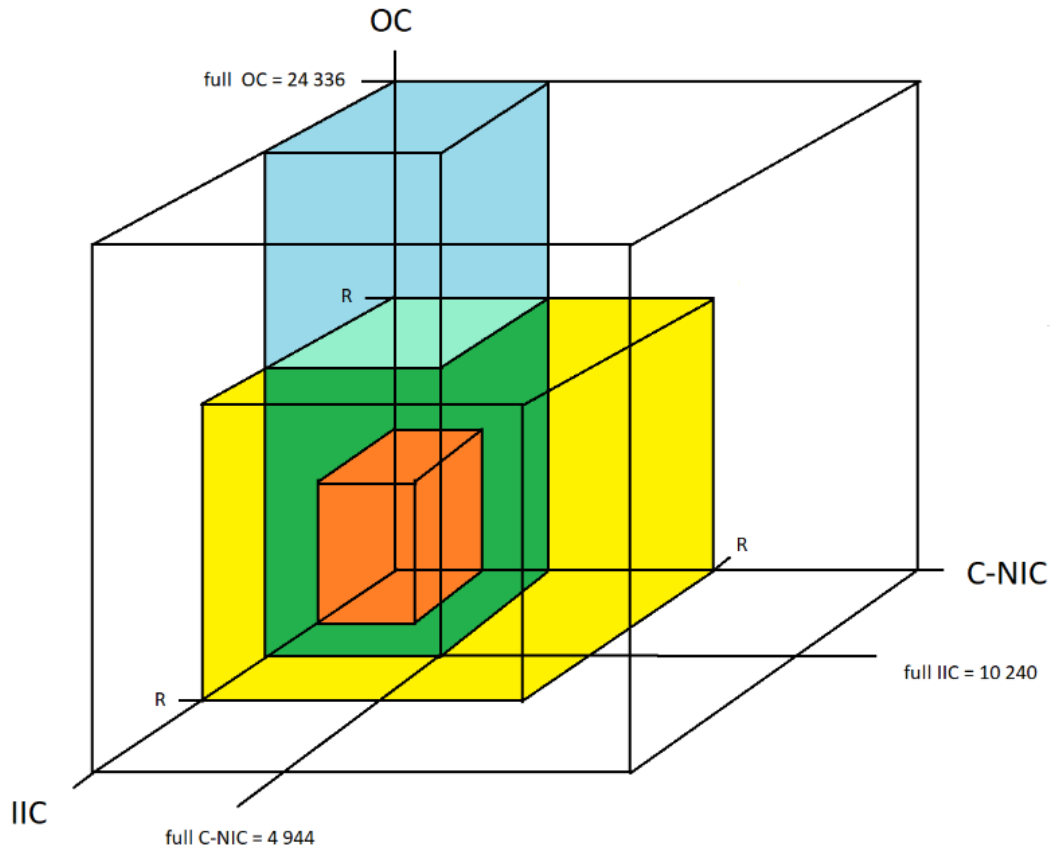

Figure 14. Geometrical illustration of parallel optimization for equation (5-1).

To find the theoretical minimum for the number of combinations required for full overall coverage. It is assumed that to minimize the required combinations, each combination must increase the OC, as it required the most combinations for full coverage. Assume that every combination includes a new output pattern, in that case 24 336 combinations would achieve full OC. Whether this can be declared as minimum for overall coverage, following statements must be true: 1) There must exists such subset that achieves full OC that also achieves full C-NIC while using 24 336 combinations. 2) There must exists a such subset that achieves full OC that also achieves full IIC with 24 336 combinations. If either statement is false, full overall coverage will require more than 24336 combinations.

For the first statement, it must be considered if there exists a such scenario, in which a unique C-NIC combination can only be found from a single output pattern, while another identical output pattern contains a different unique C-NIC combination. If such scenario occurs, there must be overlapping in output sense to be able to include all the C-NIC combinations. This is illustrated in Figure 15.

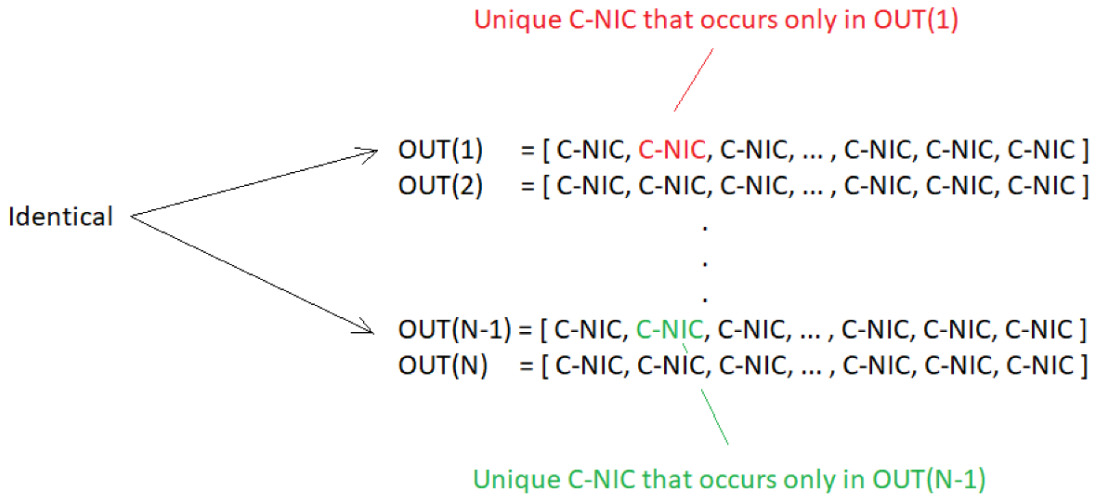

Figure 15. Characteristic preventing full overall coverage with minimum combinations required for full output coverage.

From Figure 15, consider that outputs 1 and N-1 are identical, i.e. they would get assigned a same output ID. This means that to achieve full OC with minimum combinations only one of the patterns is included, resulting that a unique C-NIC combination would not be included. One this kind of scenario is enough to prevent achieving full overall coverage with 24 336 combinations. Algorithm that checks whether such scenario occurs, in equation (5-1), is required.

A principle of such algorithm is illustrated in the Figure 16. The algorithm can be divided into six sections. First, all the valid combinations are generated. Second, the different C-NIC combinations and output patterns are enumerated. Third, an ID is assigned to every unique C-NIC combination and output pattern. Fourth, the assigned IDs are used to map all the output patterns in which different C-NIC combinations occur. Fifth, the C-NIC combinations that occur in a single output pattern are searched. Finally, it is checked if all the selected C-NIC combinations can be included without selecting the same output ID twice, i.e. can each output ID, that contains unique C-NIC combinations, include all the unique C-NIC combinations in some of the output patterns.

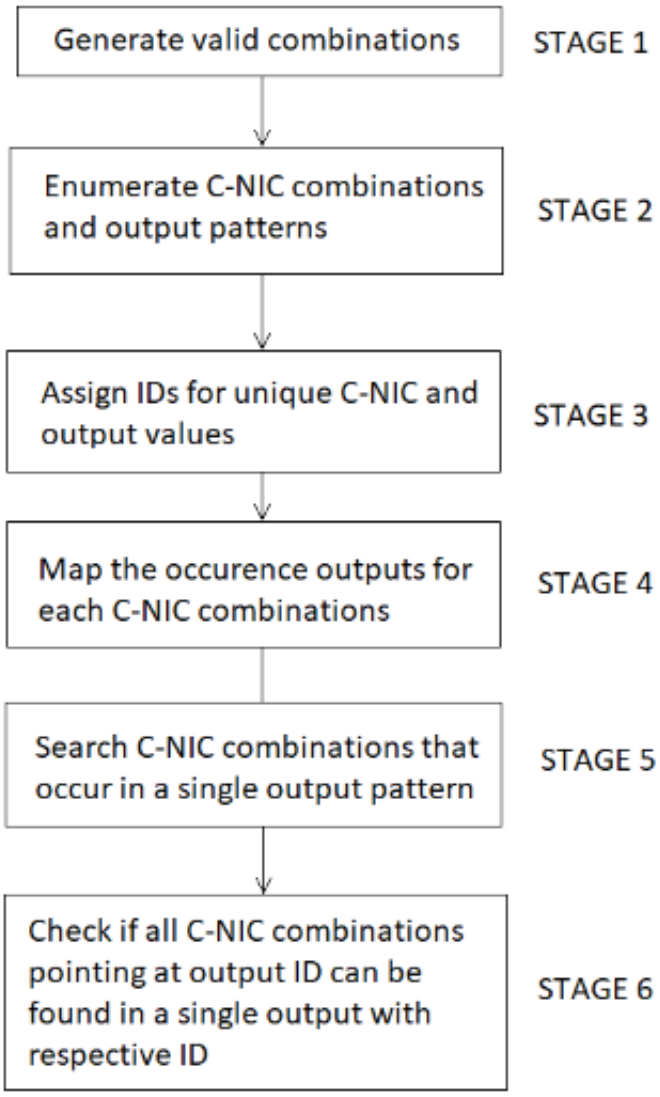

Figure 16. Principle of algorithm that checks for scenario presented in Figure 15.

The algorithm, illustrated in the Figure 16, was implemented with python. After running the algorithm, it was found out that there were total of 21 output IDs that contained unique C-NIC values, five of which contained C-NIC values that could not be found in a single output pattern. Thus, statement one can be declared as false, resulting that full overall coverage will require more than 24 336 combinations.

To determine the number of combinations that are required to achieve a full combined OC and C-NIC for the equation (5-1), another algorithm was created. The algorithm searches the minimum increment for the number of output patterns required to include all the C-NIC combinations, while maintaining full OC. The number of required combinations to achieve the full overall coverage increases by at least 17 combinations due to OC and C-NIC relation. Hence, full OC with full C-NIC coverage will require at least 24 353 combinations. The algorithm is presented in Figure 17.

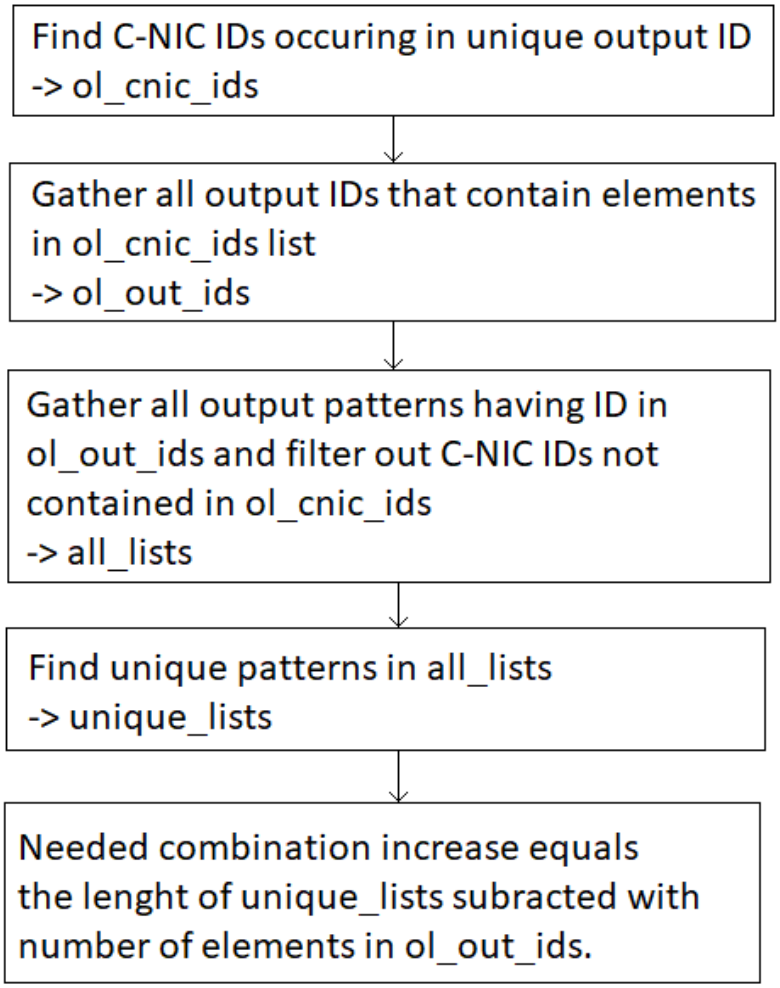

Figure 17. Algorithm that determines combination increment required to achieve a full OC/C-NIC coverage.

Next, it is considered if a subset, that achieves full OC and C-NIC coverages, achieves a full IIC too. If full OC and C-NIC is assumed, it is guaranteed that at least: $n_{\mathrm{NB}}^{(i_0)}$, $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, $N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$, $f_{\mathrm{NB,hop}}^{\mathrm{DL}}$, $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$, and $N_{\mathrm{NB}}^{\mathrm{DL}}$ achieve all their possible values. Full OC guarantees all values for $n_{\mathrm{NB}}^{(i_0)}$, $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, $N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$, $f_{\mathrm{NB,hop}}^{\mathrm{DL}}$, and $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$ because they directly affect the shape of the pattern. $n_{\mathrm{NB}}^{(i_0)}$ defined the starting narrowband, $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ defined the hopping interval in time domain. $N_{\mathrm{NB,hop}}^{\mathrm{ch,DL}}$ determined the number of different narrowbands used for frequency hopping. $f_{\mathrm{NB,hop}}^{\mathrm{DL}}$ determined the size of the hop in frequency domain. $N_{\mathrm{abs}}^{\mathrm{PDSCH}}$ defined the length of the hopping pattern. Full C-NIC guarantees all values for $N_{\mathrm{NB}}^{\mathrm{DL}}$ because it was directly mapped as a NIC term in in equation (5-8). However, full C-NIC and OC does not guarantee all i_0 values. In OC sense eight consecutive i_0 values were enough to produce all the output patterns. In C-NIC sense, i_0 does not affect the equation states. This can be proved by direct manipulation of the respective NIC term equation, i.e. the equation (5-4), defining the term $T_{1,1}$. First, the values of i, i_4, and j_0 are inserted into equation (5-4), which result into expression illustrated in equation (5-9)

$$T_{1,1} = \left\lfloor \frac{i_0 + N_{\text{abs}}^{\text{PDSCH}} - 1}{N_{\text{NB}}^{\text{ch,DL}}} - \left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor \right\rfloor \quad (5-9)$$

By denoting $N_{\text{abs}}^{\text{PDSCH}} - 1 = N$, equation (5-9) can be expressed as in equation (5-10).

$$T_{1,1} = \left\lfloor \frac{i_0 + N}{N_{\text{NB}}^{\text{ch,DL}}} - \left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor \right\rfloor \quad (5-10)$$

By expanding the term $\left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor$ by $N_{\text{NB}}^{\text{ch,DL}}$, the equation becomes as in equation (5-11).

$$T_{1,1} = \left\lfloor \frac{i_0 + N}{N_{\text{NB}}^{\text{ch,DL}}} - \frac{\left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}}}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor \quad (5-11)$$

Which can be reduced into equation (5-12) form.

$$T_{1,1} = \left\lfloor \frac{1}{N_{\text{NB}}^{\text{ch,DL}}} \left(N + \left(i_0 - \left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right) \right) \right\rfloor \quad (5-12)$$

From the equation (5-12), the term $\left(i_0 - \left\lfloor \frac{i_0}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right)$ can achieve only values: $0 \ldots N_{\text{NB}}^{\text{ch,DL}} - 1$. Which can be seen if it is denoted that $i_0 = N_{\text{NB}}^{\text{ch,DL}} + b$, where b is any positive integer number. After replacing i_0 with $N_{\text{NB}}^{\text{ch,DL}} + b$, equation (5-12) becomes as equation (5-13), which is further simplified into different forms as presented in the equations (5-14), (5-15), and (5-16).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\text{NB}}^{\text{ch,DL}}} \left(N + \left(N_{\text{NB}}^{\text{ch,DL}} + b - \left\lfloor \frac{N_{\text{NB}}^{\text{ch,DL}} + b}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right) \right) \right\rfloor \quad (5-13)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\text{NB}}^{\text{ch,DL}}} \left(N + \left(N_{\text{NB}}^{\text{ch,DL}} + b - \left\lfloor 1 + \frac{b}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right) \right) \right\rfloor \quad (5-14)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\text{NB}}^{\text{ch,DL}}} \left(N + \left(N_{\text{NB}}^{\text{ch,DL}} + b - N_{\text{NB}}^{\text{ch,DL}} - \left\lfloor \frac{b}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right) \right) \right\rfloor \quad (5-15)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\text{NB}}^{\text{ch,DL}}} \left(N + \left(b - \left\lfloor \frac{b}{N_{\text{NB}}^{\text{ch,DL}}} \right\rfloor N_{\text{NB}}^{\text{ch,DL}} \right) \right) \right\rfloor \quad (5-16)$$

Now, considered two scenarios: $b < N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, and $b >= N_{\mathrm{NB}}^{\mathrm{ch,DL}}$. If $b < N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ equation (5-16) becomes as in equation (5-17).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} (N + b) \right\rfloor, \qquad \text{where } 0 \leq b < N_{\mathrm{NB}}^{\mathrm{ch,DL}} \tag{5-17}$$

Now if $b >= N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, equation (5-17) becomes as (5-18).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + \left(b - \left\lfloor 1 + \frac{b - N_{\mathrm{NB}}^{\mathrm{ch,DL}}}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \right\rfloor N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right) \right\rfloor \tag{5-18}$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + \left(b - N_{\mathrm{NB}}^{\mathrm{ch,DL}} - \left\lfloor \frac{b - N_{\mathrm{NB}}^{\mathrm{ch,DL}}}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \right\rfloor N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right) \right\rfloor \tag{5-19}$$

Now if $b - N_{\mathrm{NB}}^{\mathrm{ch,DL}} = c$, the equation (5-19) becomes as the equation (5-20), which is of same form as the equation (5-16).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + \left(c - \left\lfloor \frac{c}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \right\rfloor N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right) \right\rfloor \tag{5-20}$$

Again, the equation (5-20) can be divided into two sections by considering $c < N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, and $c >= N_{\mathrm{NB}}^{\mathrm{ch,DL}}$. If $c < N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, the equation (5-20) becomes as in the equation (5-21).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} (N + c) \right\rfloor, \qquad \text{where } N_{\mathrm{NB}}^{\mathrm{ch,DL}} \leq c < 2N_{\mathrm{NB}}^{\mathrm{ch,DL}} \tag{5-21}$$

And if $c >= N_{\mathrm{NB}}^{\mathrm{ch,DL}}$, equation (5-20), can be expressed as in equation (5-22).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + \left(c - N_{\mathrm{NB}}^{\mathrm{ch,DL}} - \left\lfloor \frac{c - N_{\mathrm{NB}}^{\mathrm{ch,DL}}}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \right\rfloor N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right) \right\rfloor \tag{5-22}$$

This process can be continued infinitely, i.e. next step would be denoting $c - N_{\mathrm{NB}}^{\mathrm{ch,DL}} = d$ and repeating the process results into expression in equation (5-23).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} (N + d) \right\rfloor, \qquad \text{where } 2N_{\mathrm{NB}}^{\mathrm{ch,DL}} \leq d < 3N_{\mathrm{NB}}^{\mathrm{ch,DL}} \tag{5-23}$$

The relation to i_0 can be seen if the values of d, c, and, b, are inserted into equation (5-24). The insertion processes are presented in equations (5-24) to (5-26), resulting in a generic expression of $T_{1,1}$ presented in equation (5-27).

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + c - N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right\rfloor \text{ where, } \quad N_{NB}^{\mathrm{ch,DL}} \leq c < 2N_{\mathrm{NB}}^{\mathrm{ch,DL}} \qquad (5-24)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + b - 2N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right\rfloor \text{ where, } \quad 2N_{NB}^{\mathrm{ch,DL}} \leq b < 3N_{NB}^{\mathrm{ch,DL}} \qquad (5-25)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + i_0 - 3N_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right\rfloor \text{ where, } \quad 3N_{NB}^{\mathrm{ch,DL}} \leq i_0 < 4N_{NB}^{\mathrm{ch,DL}} \qquad (5-26)$$

$$T_{1,1} = \left\lfloor \frac{1}{N_{\mathrm{NB}}^{\mathrm{ch,DL}}} \left(N + i_0 - aN_{\mathrm{NB}}^{\mathrm{ch,DL}} \right) \right\rfloor \qquad (5-27)$$

where,

$$aN_{\mathrm{NB}}^{\mathrm{ch,DL}} \leq i_0 < (a+1)N_{\mathrm{NB}}^{\mathrm{ch,DL}}, \text{and } a \in \mathbf{Z}^+$$

Given that the maximum value of the parameter $N_{\mathrm{NB}}^{\mathrm{ch,DL}}$ was eight, all the C-NIC combinations can be included with eight consecutive values of i_0. Thus, full OC, and C-NIC does not guarantee full IIC, but it must be verified separately.

6 OPTIMAL SUBSET SEARCH

In this chapter, various algorithms for optimal subset search are introduced and compared. The chapter is divided into three sections. In the first section, an optimal subset is searched in a scenario where there are no resource restrictions. In the second section, an optimal subset is searched in a scenario when a resource limit exists. Finally, test resource utilization performances of a various subset sizes are presented, and an approximation of the optimal subset performance is plotted as a function of the subset size R.

6.1 Unlimited Resources

An unlimited resource scenario is required to determine an approximation of the minimum number of resources$s, R_{\min}$, required for the full overall coverage, which is used for subset performance calculations. GA and WGA are the two chosen algorithms used to approximate the $R_{\min}$. BGA is not feasible for this case as step size two would already cause the number of subsets to become nCr(2, 370 176), i.e. 68 514 950 400, which is not within feasible range.

6.1.1 Greedy algorithm

The GA requires the knowledge of all the outputs, C-NIC combinations, and input parameters contained in the selected CS. The GA is initialized so that all the contained elements in the CS are declared as missing. At the start of each iteration, a temporary subset is initialized as empty set. Next, all the elements in the current combination that can be found from the respective missing set, are added to the temporary set. At the end of the iteration, algorithm decides whether the temporary data is included to the final subset. The decision is made based on a threshold of the current iteration. The threshold is the required number of missing elements that must be found from the handled iteration. If found elements is equal or larger than threshold, temporary data is included in the final subset, and the found elements are removed from the missing elements. Otherwise, the temporary data is discarded, and the algorithm moves to the next iteration. The "Greediness" of the algorithm was implemented by assigning an initial threshold of 40, which is the maximum possible new elements in a combination, i.e. a new output, 32 new C-NIC, and seven new input parameters. Threshold is decreased by one after all combinations have been iterated with assigned threshold and decreased until all missing elements are found. The obtained value of the $R_{\min}$ with GA was 24 367 combinations.

6.1.2 Weighed greedy algorithm

In WGA, different weight factors can be assigned for the output, C-NIC, and input parameters. In GA, C-NIC combinations were dominant due to the large numbers of C-NIC combinations in a single combination, i.e., combinations with large $N_{\text{abs}}^{\text{PDSCH}}$ values were initially selected. To shift the stress more to new output combinations, weight of 10 was assigned to output and initial threshold was increased to 49, which is the new maximum increase after the added weight is considered. With new the weight of 10 for outputs, WGA favours new output combinations after large numbers of new C-NIC combinations can no longer be found. The inputs weight was

not changed, as a small number of input values were found relatively quickly. Achieved $R_{\min}$ with WGA was 24 365 which is marginally better than what GA achieved.

6.2 Limited Resources

Finding an optimal subset with limited resources is desired as unlimited resources cannot be assumed in practice. In limited resource scenario, target is to maximize the overall coverage with a fixed number of testing resources. As initial approach a brute force search algorithm is considered.

6.2.1 Brute force search

Brute force search is an algorithm that iteratively finds the subset that achieves the best coverage. Brute force search is guaranteed to arrive at the optimal solution because it checks all the possible solutions and selects the best one from them. Figure 18 illustrates a brute force algorithm. First, the algorithm fills the subset with any combinations until the resource limit is met. Next, the algorithm iterates through the remaining combinations and checks if the overall coverage can be increased by replacing any of the included combinations with a non-included combination. If an unhandled combination cannot increase coverage, the combination is discarded, and the algorithm moves to the next combination. Conversely, if the coverage can be increased, the algorithm performs such replacement operation that results in the largest increase in the overall coverage. Algorithm continues until there are no more unhandled combinations, i.e. the coverage cannot be increased further by replacing combinations. Table 15 presents the rough estimates of possible combinations for included subset with different resource limits.

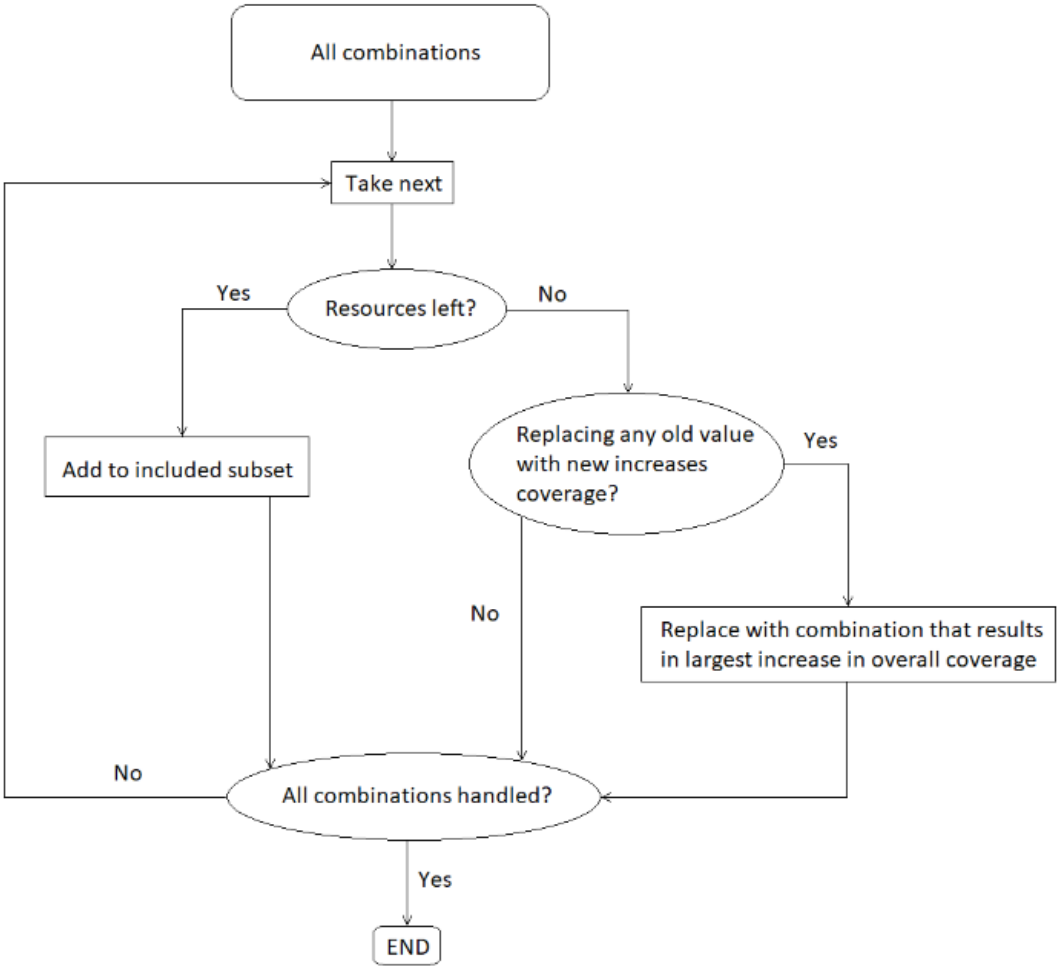

Figure 18. Algorithm that finds optimal subset with limited resources.

Table 15. The number of possible ways to select subset from the combination space of 370 176 combinations with various resource limits.

Resource limit	Combinations
10	10e49
100	10e398
1000	10e3000
10000	10e19965

Although, the brute force algorithm does not go through all possible combinations, Table 15 only illustrates the exponentially increasing complexity of the algorithm. In practice, algorithm performs comparisons between included and new potential combinations, but going through all combinations is not required. After one combination is handled, it can be forgotten, because every included combination yields better result than any previously handled. This means that the number of comparison operations, denoted as N_{comp}, required to achieve optimal subset is linearly proportional combination space size CS and included subset size R as defined in equation (6-1).

$$N_{\mathrm{comp}} = (\mathrm{CS} - R)R \qquad (6-1)$$

This can be justified by considering that the first R combinations are included without comparisons, as the resource limit R is not exceeded, reducing the CS to $CS - R$ combinations. After that, each of the remaining $CS - R$ combinations must be compared to each of the R included combinations. Given that each of the $CS - R$ combinations must be compared once to every combination in current subset, size of which is a constant R after the subset is initially filled. Therefore, the total number of comparisons must be equal to $(CS - R)R$, in order to complete the brute force search.

Table 16 presents the required comparisons, average time required per comparison and estimated total time to complete the brute force search with various resource limits for combination space of size 370 176. Time per comparison is determined by executing the algorithm partially while measuring the time taken to execute and counting the comparison operations. The average time per comparison is calculated by dividing the total time by the number of comparisons, counted from the partial execution.

Table 16. Number of comparisons, time per comparison and execution time estimate for optimization algorithm for limited resources.

Resource limit	Comparisons	Time per Comparison	Total time estimate
10	3 701 660	0.916 ms	56 min 31 s
100	37 007 600	1.474 ms	14 h 49 min 7 s
1000	369 176 000	63.98 ms	273 d 9 h 50 min
10000	3 601 760 000	5.420 s	617 y 8 d 17 h

From Table 16, the number of comparisons grows linearly with respect to the resource limit. However, increasing the subset size exponentially increases the required time per comparison, exponentially increasing the total time. The exponential increase in the execution time results that using brute force algorithm is not feasible for resource limits much larger than 100. The exponential behaviour of time per comparison is caused by the larger list size of final subset, resulting that all operations required for comparison, i.e. adding, removing, and finding parameters from lists take longer to execute.

6.2.2 Sub-Group Division

To achieve feasible execution time for larger subset sizes, the exponential behaviour of time per comparison must be removed. To achieve that, a Sub-Group Division method (SGD) is proposed. The exponential behaviour can be removed by dividing search into sub-groups and performing brute force search for each sub-group. After each sub-group is optimized, all the resulting subsets are merged into one final subset. The number of comparisons per sub-group, denoted as $N_{\mathrm{comp,sub}}$, for N_{sub} equal sized sub-groups, is defined in equation (6-2). Total comparisons required for equal sized sub-groups is defined in equation (6-3).

$$N_{\mathrm{comp,sub}} = \left(\frac{CS}{N_{\mathrm{sub}}} - \frac{R}{N_{\mathrm{sub}}}\right)\frac{R}{N_{\mathrm{sub}}} \qquad (6-2)$$

$$N_{\mathrm{comp}} = N_{\mathrm{comp,sub}} N_{\mathrm{sub}} \qquad (6-3)$$

Total number of comparisons for varying sized sub-groups is presented in equation (6-4). Where, $N_{\text{comp,sub},n}$ is the number of comparisons required for the n^{th} sub-group. $N_{\text{comp,sub},n}$ is defined in the equation (6-5), where CS_n is the size of the combination space of the n^{th} sub-group and R_n is the number of resources allocated to the n^{th} sub-group.

$$N_{\text{comp}} = \sum_{n=1}^{N_{\text{sub}}} N_{\text{comp,sub},n} \quad (6-4)$$

$$N_{\text{comp,sub},n} = (CS_n - R_n)R_n \quad (6-5)$$

For example, if one wishes to find an optimal subset of size 1 000 from a combination space of 370 176 combinations. The combination space could be divided into 100 sub-groups, i.e. $N_{\text{sub}} = 100,$ with evenly distributed resources. The combination space and resources per sub-group becomes: CS/N_{sub}=3 702 and R/N_{sub}=10. The numerical value for the required comparisons for each sub-group is calculated in equation (6-6). The total number of comparisons for example scenario is calculated in equation (6-7).

$$N_{\text{comp,sub}} = (3702 - 10)10 = 36\,920 \quad (6-6)$$

$$N_{\text{comp}} = N_{\text{comp,sub}}N_{\text{sub}} = 3\,692\,000 \quad (6-7)$$

Table 17, presents estimated execution times with different resource limits when SGD is used. Time per comparison for resource limit of 10 as assumed to be equal as in Table 15. In Table 17, the size of the sub-groups is assumed to be a constant.

Table 17. Number of comparisons, timer per comparison and execution time estimate when SGD is used

Resource limit	Number of sub-groups N_{sub}	Comparisons	Time per Comparison	Total time estimate
10	1	3 701 660	0.916 ms	56 min 31 s
100	10	3 700 800	0.916 ms	56 min 30 s
1000	100	3 692 000	0.916 ms	56 min 22 s
10000	1000	3 610 000	0.916 ms	55 min 7 s

From Table 17, the SGD reduces total execution time estimate, instead of exponentially increasing it, as the subset size grows larger. This is a very encouraging result as it means that the execution time is linearly proportional to the number of comparisons, which in turn are linearly proportional to total size of the combination space.

However, SGD does not guarantee optimality as there are no knowledge of the already found elements between the sub-groups, potentially resulting in unnecessary overlapping in coverage sense. Also, the optimal resource allocation among sub-groups might differ from equal allocation. If the sub-groups were independent in coverage sense, there would be no overlapping, but optimal resource allocation would still not be known. For equation (5-1), independency between sub-groups cannot be achieved as same C-NIC combinations, and inputs, can occur in different outputs. Some of the overlapping can be avoided by choosing sub-

groups with respect to OC and providing information about already found elements from previous sub-group to the next sub-group, which is discussed more in section 6.2.3.

In SGD, the sub-groups should be chosen so that each output ID can be found only from a single sub-group. Assigned resources per group should be chosen with respect to sub-group sizes, to allow equal potential for optimization for each sub-group. Two approaches to select sub-groups for SGD are proposed. In first approach, all the output IDs are enumerated, and the sub-groups are formed so that each sub-group includes an equal range of enumerated output IDs. This will possibly cause un-even subset sizes, because some of the output IDs might be more frequent than others. Hence, the resources must be assigned afterwards with respect to sub-groups size so that the ratio of resources to combination space size is equal for all the sub-groups. In second approach, equal number of resources are assigned for each sub-group, and the sub-groups are selected to be as equally-sized as possible while including each output ID in only a single sub-group. Figure 19, illustrates two proposed approaches for the SGD sub-group selection.

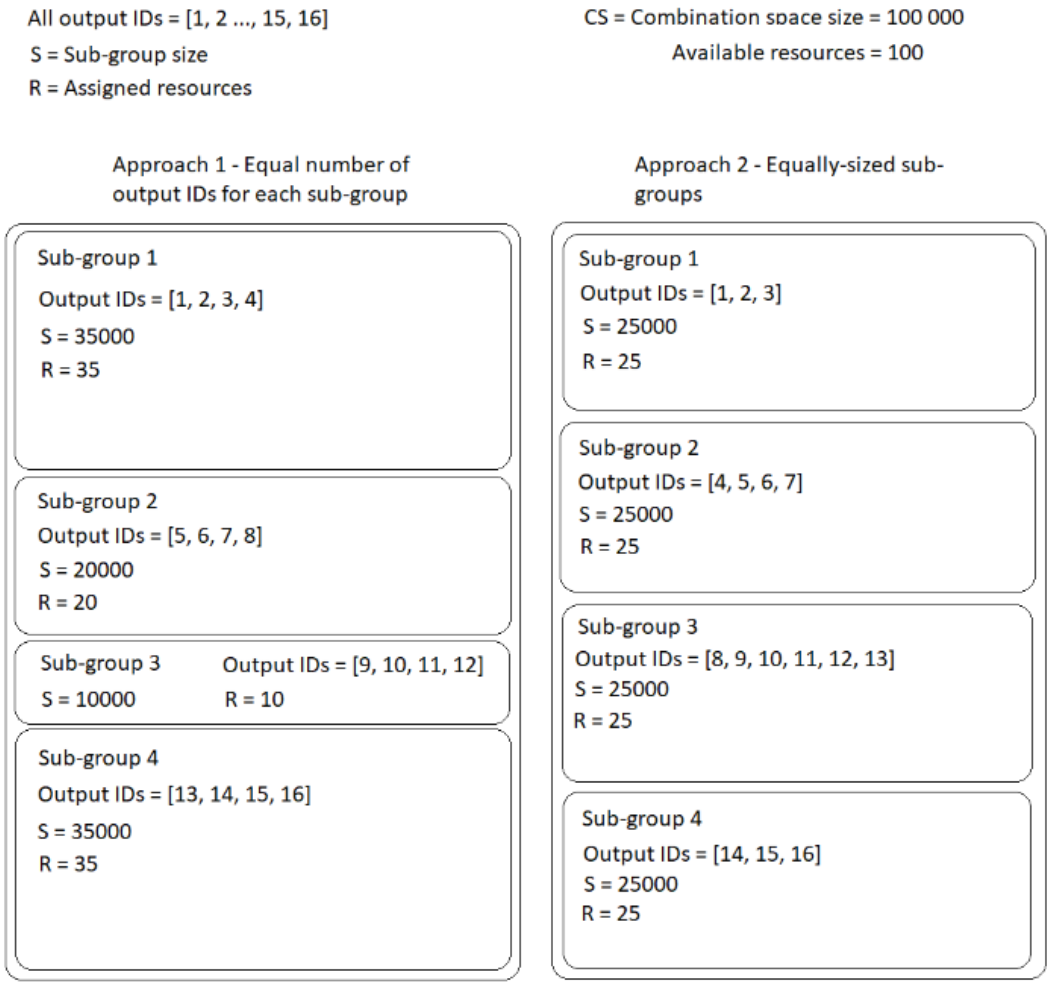

Figure 19. Two approaches for selecting sub-groups in SGD-method.

Due to the exponential behaviour of the execution time of brute force search, the approach 2 is preferred because the resources for the sub-groups can be assigned to be constant. Equal-sized sub-groups guarantees a feasible execution time. One larger sub-group is enough to result to longer execution time. This can be illustrated by calculating a squared sum of sub-group sizes (squared to mimic exponentiality). Squared sum can be used to compare the execution times of

SGD approaches. For example, the squared sum for SGD approach 1, denoted as S_1, and SGD approach 2, denoted as S_2, for sub-group sizes presented in Figure 19, are presented in equations (6-8) and (6-9).

$$S_1 = \sum_{n=1}^{4} R_n^2 = 35^2 + 20^2 + 10^2 + 35^2 = 2950 \quad (6-8)$$

$$S_2 = \sum_{n=1}^{4} R_n^2 = 25^2 + 25^2 + 25^2 + 25^2 = 2500 \quad (6-9)$$

From equation (6-8) and (6-9) results, execution time for SGD approach 1 takes roughly 20% longer compared to SGD approach 2 with equal subset sizes. Note that the exponential behaviour is stronger for bigger the sub-group sizes. For example, from Table 16, 10-fold increase in subset size from 100 to 1 000 resulted in ~440-fold increase in execution time, and further 10-fold increase in subset size from 1 000 to 10 000 resulted in ~820-fold increase in execution time.

Approach 1 can be modified so that the enumerated output IDs per sub-groups are divided until the maximum assigned resources of all the sub-groups are below a desired threshold. Conversely, sub-groups can be merged if some of the sub-groups are so small that no resources would be assigned for them, or to achieve pre-determined minimum sub-group size. In Figure 18 scenario, if the desired upper limit for R is set to 25, for approach 1, sub-groups 1 and 4 would be further divided into four sub-groups including output IDs: {1, 2}, {3, 4}, {13, 14}, and {15, 16} respectively, which would result that all the sub-groups sizes are below the desired threshold. This will result in larger number of sub-groups and faster execution time, but potentially to worse performance.

Both approaches for SGD were tested with python code that executes algorithm presented in Figure 18, for each sub-group. Results for both approaches, with various resource limits, are gathered in Table 18 and Table 19. Test was done with combination space of size 100 000.

Table 18. Results for SGD approach 1 for combination space of size 100 000

Resource limit	Approach 1			
	N_{sub}	Comparisons	Time	Coverage
10	1	999 900	15 min 42 s	33.27%
100	13	936 228	18 min 43 s	53.48%
1 000	154	947 160	6 min 6 s	70.02%
10 000	1181	1 537 620	9 min 57 s	93.54%

Table 19. Results for SGD approach 2 for combination space of size 100 000

Resource limit	Approach 2			
	N_{sub}	Comparisons	Time	Coverage
10	1	999 900	15min 19 s	33.27%
100	10	999 000	14min 47 s	49.41%
1 000	100	990 000	9min 38 s	68.60%
10 000	1000	900 000	7min 34 s	99.82%

When the execution times of the approaches 1 and 2 are compared, from Table 18 and Table 19, approach 2 appears to be faster than approach 1. In the approach 1, the number of sub-

groups increase due to the splitting of larger combination spaces. In approach 2, the number of comparisons and execution time decreases when resource limit increases. In addition, it is noticed that the number of comparisons does not directly correlate with the execution time because comparisons are not equal in complexity as it is described next.

There are a few factors that have an impact on the execution time. Such factors include: the number of sub-groups, combination space size, assigned resources, and the average size of included combinations, i.e. the average number of different outputs, C-NICs, and input parameters handled within the sub-groups.

The number of sub-groups directly controls the average sub-group combination space size and the average resources per sub-group. Smaller sub-groups are faster to optimize because both smaller CS and less resources linearly decreased the required comparisons. Further, less resources also reduced time required per comparison. Reduction in time per comparison is due to the smaller number of primitive operations required for a single comparison. A primitive operation means a single add operation, for a combination, that consist of appending all the missing elements, that were found from the combination, to found elements. For less resource, the list sizes, used to store the found elements, are smaller and it is faster to check if an element is found from a small list of values. For same reasons, larger sub-groups are significantly slower to optimize.

From Table 18, and Table 19, the difference in execution time between SGD approaches, due to uneven sub-group size is visible in the scenario where R=100. Closer look to this behaviour was done by counting the number of primitive operations, the average number of outputs and C-NIC combinations included in the optimized subset during the sub-group optimization, for each sub-group. Table 20 summarizes the results for SGD approach 1, and Table 21, for SGD approach 2. Notation used in the Table 20, and Table 21 is as follows: n is the index of sub-group, CS_n is the CS of n^{th} sub-group, R_n is the resources allocated for n^{th} sub-group, T_n is the time taken to optimize n^{th} sub-group, $N_{\text{add},n}$ is the number of primitive operations required for n^{th} sub-group, $\mu_{\text{out},n}$ is the average number of outputs included in the optimized subset during optimization of n^{th} sub-group, and $\mu_{\text{cnic},n}$ is the average number of C-NIC combinations included in the optimized subset during the optimization of n^{th} sub-group.

Table 20. Sub-group information for approach 1 when R=100 and CS = 100 000

	n	CS_n	R_n	T_n	$N_{\text{add},n}$	$\mu_{\text{out},n}$	$\mu_{\text{cnic},n}$
	13	2 336	3	2.57 s	42 145	1.81	46.69
	7	3 898	4	6.77 s	124 757	2.99	68.02
	9	4 128	4	7.03 s	132 102	2.99	67.6
	8	4 598	5	9.73 s	229 901	3.98	72.41
	6	4 902	5	11.9 s	245 113	3.98	79.28
	5	6 734	7	29.4 s	659 701	5.98	112.19
	4	7 248	7	30.5 s	710 129	5.95	106.23
	2	7 656	8	44.2 s	979 501	6.98	123.27
	11	8 938	9	59.7 s	1 447 307	6.24	133.08
	1	8 758	9	62.5 s	1 418 117	7.61	131.06
	12	12 988	12	171 s	3 738 881	7.85	178.82
	10	13 098	13	308 s	4 424 381	10.23	236.29
	3	14 716	14	367 s	5 764 968	12.57	219.45
Total	13	100 000	100	1 123 s	19 917 003	-	-

Average	-	7 692	7.69	86.4 s	1 991 700	9.06	166.37

Table 21. Sub-group information for approach 2 when R=100 and CS = 100 000

	n	CS_n	R_n	T_n	$N_{\text{add},n}$	$\mu_{\text{out},n}$	$\mu_{\text{cnic},n}$
	3	10 000	10	71.2 s	1 998 956	8.87	134.88
	1	10 000	10	78.2 s	1 999 000	8.72	142.05
	8	10 000	10	79.0 s	1 998 934	8.2	137.06
	10	10 000	10	79.3 s	1 999 077	8.35	134.07
	2	10 000	10	79.7 s	1 999 055	8.56	139.77
	5	10 000	10	86.2 s	1 999 099	8.81	141.3
	4	10 000	10	89.0 s	1 998 857	8.3	154.82
	7	10 000	10	90.8 s	1 998 989	8.93	149.16
	6	10 000	10	93.1 s	1 999 044	8.42	153.69
	9	10 000	10	96.7 s	1 999 099	8.89	161.05
Total	10	100 000	100	887 s	19 990 110	-	-
Average	-	10 000	10	88.7 s	1 999 011	8.94	160.97

By comparing results between Table 20 and Table 21, approach 2 was faster than approach 1, despite approach 1 had less average resources per sub-group (more sub-groups for same CS). In approach 1, the uneven sub-group sizes caused that the largest three sub-groups formed 75% of the total optimization time, although most were smaller than in approach 2. The difference comes mainly from the $\mu_{\text{cnic},n}$ quantity. The three largest sub-groups had the largest average number of C-NIC combinations included during the algorithm. More C-NIC combinations requires more primitive operations and longer execution time for primitive operations (due to longer lists), which also explains the exponential behaviour for execution time. In approach 2, the total execution time is distributed more evenly among the sub-groups as the $\mu_{\text{cnic},n}$ was roughly equal for every sub-group. Although, the correlation of the $\mu_{\text{cnic},n}$ to the execution time can be observed from the approach 2 as well.

In conclusion, uneven sub-groups do not directly increase the execution time. Instead, the average number of included C-NIC combinations during the optimization, which is potentially larger when more resources are available, had largest impact on the sub-group execution time.

6.2.3 Recursive information for SGD

The downside of both introduced SGD approaches was that they created sub-groups with only independent output IDs. Hence, overlapping between the sub-groups could occur in C-NIC and IIC sense. C-NIC and IIC overlapping can decrease the algorithm performance because a single C-NIC could occur in multiple output IDs. Algorithm will select combinations so that the sub-group coverage is maximized, but the selected sub-group subsets might not be optimal if their aggregation is considered. Given that e.g. a set of different C-NIC combinations were included in first sub-group and next sub-group has no information about those C-NIC combinations, the second sub-group ignores the previously selected C-NIC combinations and just maximizes the sub-group coverage, not the CS coverage, and might select the same C-NIC combinations again, even if there would exist a better option with respect to whole CS. One way to counter the coverage overlapping is to modify the SGD algorithm so that after each sub-group is optimized,

knowledge of previously included C-NIC combinations and input parameters is passed to the next sub-group. With the added information, the next sub-group will only select a combination with already handled C-NIC combinations, or inputs, if the overall coverage of the whole CS is improved. Modified algorithm is referred as Recursive Sub-Group Division (R-SGD), or to denote R-SGD with a specific SGD approach, R-SGD-1 for approach 1, and R-SGD-2 for approach 2, notation is used. In Table 22, the performances of R-SGD and SGD are compared.

Table 22. Performance comparison of R-SGD and SGD

Resource limit	R-SGD-1	R-SGD-2	SGD-1	SGD-2
10	29.22%	29.22%	29.22%	29.22%
100	52.49%	51.72%	42.62%	39.86%
1000	66.24%	66.35%	65.51%	60.70%
10000	77.80%	78.31%	77.81%	78.82%

From the Table 22, there was no superior algorithm that would achieve best performance at all resource limits. Based on the result presented in the Table 22, the optimization algorithm should be chosen with respect to available resources, e.g. R-SGD achieves better result for small resource limits, whereas the original SGD performed marginally better on larger sets. R-SGD is also slower than original SGD due to the increasing size of recursive information, accumulating from the previous sub-groups.

6.2.4 Greedy algorithm for limited resources

SGD method with recursive information is somewhat time taking algorithm. WGA and GA are much lighter algorithms. In this section, optimization for limited resources scenario is attempted with GA and WGA. The results of GA and WGA compared to the R-SGD-1, which resulted to best coverage for smaller subsets, are presented in table 23.

Table 23. Performance comparison of WGA and GA

Resources	WGA	GA	R-SGD-1
10	25.74%	25.75%	29.22%
100	47.31%	47.31%	52.49%
1000	67.21%	67.99%	66.24%
10000	79.54%	80.32%	77.80%

From Table 23, GA is better for larger subsets while R-SGD-1 remains the best algorithm for smaller subsets. WGA was worse than GA, but better than R-SGD-1 with larger subsets.

6.2.5 R-SGD with GA and WGA as utility

R-SGD had the best performance for small subsets, whereas GA yielded better results as available resources increased. Question is: what if these algorithms were combined? R-SGD used the overall coverage directly as a utility function, which naturally maximized the overall coverage, whereas GA increased the overall coverage only non-directly by including missing

elements. In this section, the performance of various combinations of R-SGD and GA are compared.

Combination of R-SGD and GA were implemented on top of the R-SGD algorithm. The sub-groups were formed as before, but this time the calculation of overall coverage for each iteration was replaced with WGA, i.e. instead of calculating the overall coverage, the algorithm included as many missing elements with given resources as possible. WGA was implemented as it could easily be reduced to GA by assigning weight of one for each missing element. Both SGD approaches with both WGA and GA were tested. The weights for WGA were same as in unlimited resource scenario, presented in section 6.1.2, i.e. ten for outputs and one for everything else. Results for the combined algorithms are presented in Table 24.

Table 24. Performance comparison of R-SGD-WGA and R-SGD-GA

Resource limit	R-SGD-1-WGA	R-SGD-2-WGA	R-SGD-1-GA	R-SGD-2-GA
10	25.74%	25.74%	25.74%	25.74%
100	49.06%	40.01%	48.44%	40.01%
1000	66.50%	66.60%	66.78%	67.19%
10000	78.16%	78.95%	78.05%	79.12%

If R-SGD with GA is used, SGD approach and utility should be chosen based on the resource limit. From the Table 24 can be seen that the SGD approach 1 with WGA should be selected when resource limit is small, whereas for larger subsets the SGD approach 2 with GA provides better results. For very small subsets, there is no difference in performance because there is just a single sub-group, the complete CS. Table 25 presents the comparison of the combined algorithms to the best separate results of GA and R-SGD-1.

Table 25. Performance comparison of combined algorithms to the best results of separate algorithms

Resource limit	R-SGD-1-WGA	R-SGD-2-GA	GA	R-SGD-1
10	25.74%	25.74%	25.75%	29.22%
100	49.06%	40.01%	47.31%	52.49%
1000	66.50%	67.19%	67.99%	66.24%
10000	78.16%	79.12%	80.32%	77.80%

From the Table 25, combining the algorithms did not achieve better coverage for any of the tested resource limits. Adding GA for R-SGD resulted in a small performance gain for larger subsets. However, with a cost of significant degradation at smaller subset sizes, as summarized in Table 26 and Table 27.

Table 26. R-SGD-1 performance comparison with different utilities

Resource limits	R-SGD-1	R-SGD-1-GA		R-SGD-1-WGA	
	C	C	Difference	C	Difference
10	29.22%	25.74%	-3.48%	25.74%	-3.48%
100	52.49%	48.44%	-4.05%	49.06%	-3.43%
1000	66.24%	66.78%	+0.54%	66.50%	+0.26%
10000	77.80%	78.05%	+0.25%	78.16%	+0.36%

Table 27. R-SGD-2 performance comparison with different utilities

Resource limits	R-SGD-2	R-SGD-2-GA		R-SGD-2-WGA	
	C	C	Difference	C	Difference
10	29.22%	25.74%	-3.48%	25.74%	-3.48%
100	51.72%	40.01%	-11.71%	40.01%	-11.71%
1000	66.35%	67.19%	+0.84%	66.60%	+0.25%
10000	78.31%	79.12%	+0.81%	78.95%	+0.64%

6.3 Optimal subset performance

To calculate value for the subset performance, the system parameters: $C_{\text{inc,oc}}$, $C_{\text{inc,cnic}}$, $C_{\text{inc,par},i}$, $R_{\min}$, $\varepsilon_{\text{single}}$, $\varepsilon_{\text{full}}$, and θ, must be determined for the equation (5-1). First, the coverage increases per each found output, C-NIC combination, and input parameter: $C_{\text{inc,oc}}$, $C_{\text{inc,cnic}}$, $C_{\text{inc,par},i}$, are calculated, as defined in equations (4-33), (4-34), and (4-35). From section 5.1.5, it is known that for equation (5-1), $N_{\text{oc}} = 24336$, and $N_{\text{cnic}} = 4944$. $C_{\text{inc,oc}}$ and $C_{\text{inc,cnic}}$ for equation (5-1) are presented in equations (6-10), and (6-11).

$$C_{\text{inc,oc}} = \frac{1}{3N_{\text{oc}}} = \frac{1}{73008} \qquad (6-10)$$

$$C_{\text{inc,cnic}} = \frac{1}{3N_{\text{cnic}}} = \frac{1}{14832} \qquad (6-11)$$

Regarding the input parameters, it is known from the section 5.1.1 that the number of parameters for equation (5-1) is seven, thus, $N_{\text{par}} = 7$. The number of possible values for each input parameter $N_{\text{val,par},i}$ and the respective $C_{\text{inc,par}}$, with restricted CS, are presented in Table 28.

Table 28. $N_{\text{val,par}}$ values for all parameter of equation (5-1)

Parameter	$N_{\text{val,par}}$	$C_{\text{inc,par}} = \frac{1}{N_{\text{val,par}}} \frac{1}{N_{\text{par}}} \frac{1}{3}$
i_0	16	$\frac{1}{16}\frac{1}{7}\frac{1}{3} = \frac{1}{336}$
$n_{\text{NB}}^{(i_0)}$	16	$\frac{1}{16}\frac{1}{7}\frac{1}{3} = \frac{1}{336}$
$N_{\text{NB}}^{\text{ch,DL}}$	4	$\frac{1}{4}\frac{1}{7}\frac{1}{3} = \frac{1}{84}$
$N_{\text{NB,hop}}^{\text{ch,DL}}$	2	$\frac{1}{2}\frac{1}{7}\frac{1}{3} = \frac{1}{42}$
$f_{\text{NB,hop}}^{\text{DL}}$	16	$\frac{1}{16}\frac{1}{7}\frac{1}{3} = \frac{1}{336}$
$N_{\text{NB}}^{\text{DL}}$	5	$\frac{1}{5}\frac{1}{7}\frac{1}{3} = \frac{1}{105}$
$N_{\text{abs}}^{\text{PDSCH}}$	6	$\frac{1}{6}\frac{1}{7}\frac{1}{3} = \frac{1}{126}$

Second, the quantities of $R_{\min}$, $\varepsilon_{\text{opt,full}}$, $\varepsilon_{\text{opt,single}}$, and θ are determined. Minimum resources required for the full overall coverage was determined, in section 6.1, to be at most 24365, which is used as an approximated value for $R_{\min}$. The respective overall efficiency can be calculated

as defined in the equation (4-28). The value of the $\varepsilon_{\text{opt,full}}$ for equation (5-1) is calculated in equation (6-12):

$$\varepsilon_{\text{opt,full}} = \frac{C_{\max}}{R_{\min}} = \frac{1}{24365} \tag{6-12}$$

The maximum discrete efficiency, $\varepsilon_{\text{opt,single}}$ occurs in a scenario where all the input parameters are new, output is new, and the output pattern consists of 32 new C-NIC combinations. 32 is the longest possible output pattern with restricted CS, which can only be achieved if $N_{\text{abs}}^{\text{PDSCH}}$=32. The respective discrete efficiency can be calculated from the coverage increases per new element as presented in equations (6-13) and (6-14).

$$\varepsilon_{\text{opt,single}} = C_{\text{inc,oc}} + 32C_{\text{inc,cnics}} + \sum_{i=1}^{7} C_{\text{inc},i} \tag{6-13}$$

$$\varepsilon_{\text{opt,single}} = \frac{1}{73008} + \frac{32}{14832} + \frac{3}{336} + \frac{1}{126} + \frac{1}{105} + \frac{1}{84} + \frac{1}{42} = \frac{8458309}{131596920} \tag{6-14}$$

The value of the θ can be approximated as defined in equation (4-29), where $\varepsilon_{\text{opt,single,max}}$ is the maximum individual discrete efficiency among all the coverage metrics. For the equation (5-1), full coverage for parameter $N_{\text{NB,hop}}^{\text{ch,DL}}$ can be achieved using two resources. Hence, the maximum discrete efficiency is 0.5, which results the approximated θ is two. The actual value of θ is one, because the $\varepsilon_{\text{opt,single}}$ can only be achieved for first added resource. After the first resource is included, all the input parameters can no longer achieve a new value while maintaining output pattern length of 32, which was achieved when $N_{\text{abs}}^{\text{PDSCH}}$=32. Thus, the efficiency of the second added resource will be strictly less than the first, if it is assumed that the first one was selected optimally.

Finally, the approximations of the included outputs, $N_{\text{inc,oc}}$, C-NIC combinations, $\text{N}_{\text{inc,cnic}}$, and input parameters, $N_{\text{inc,par},i}$, are calculated as defined in the equations (4-34), (4-35), and (4-36). Quantities: $\frac{N_{\text{oc}}}{R_{\text{min,oc}}}$, $\frac{N_{\text{cnic}}}{R_{\text{min,cnic}}}$, and $\frac{N_{\text{val,par},i}}{R_{\text{min,par},i}}$, describe the average number of new elements of the added combinations. Regarding the outputs, and the input parameters, each combination can include only one new element. Therefore, $N_{\text{oc}} = R_{\text{min,oc}}$ and $N_{\text{val,par},i} = R_{\text{min,par},i}$. The average number of the C-NIC combinations per resource is slightly more complex as the output patterns can contain varying numbers of C-NIC combinations. The average number of C-NIC combinations per resource can be approximated e.g. with GA so that every C-NIC combination has a value of one, all output and input values have a weight of zero. GA with mentioned weights will only include all C-NIC combinations while ignoring outputs and inputs. After running the GA with mentioned weights, the approximation for $R_{\text{min,cnic}}$ is 227. Resulting approximation formulas for $N_{\text{inc,oc}}$, $N_{\text{inc,cnic}}$, and $N_{\text{inc,par},i}$ for equation (5-1), are presented in equations (6-15), (6-16), and (6-17).

$$N_{\text{inc,oc}}(R) \approx \min(R, 24336) \tag{6-15}$$

$$N_{\text{inc,cnic}}(R) \approx \min\left(\frac{4944}{227}R, 4944\right) \tag{6-16}$$

$$N_{\text{inc},i}(R) \approx \min\left(R, N_{\text{val},i}\right) \tag{6-17}$$

Now, the theoretical optimal values can be approximated for equation (5-1), which can be used to plot the absolute performance of an optimal subset at various subset sizes R. Figure 20, presents the approximation of the optimal efficiency and performance for the equation (5-1) as a function of subset size R. Figure 21, presents zoomed plot in the proximity of the maximum absolute performance.

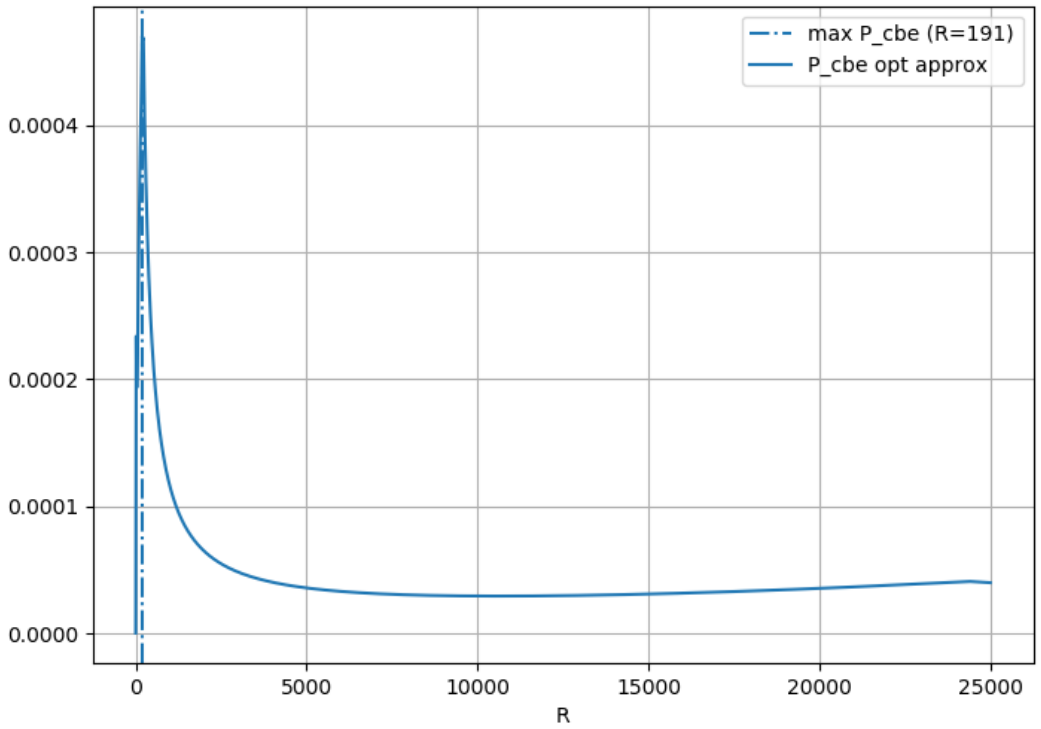

Figure 20. Approximation of the performance of an optimal subset as a function of subset size R.

From the Figure 20, at first the performance increases rapidly as the subset size R grows. Maximum performance is achieved at R=191, after which the performance begins to decrease. At roughly R=10 000, performance starts to increase slowly, until $R_{\min}$ resources are included, after which the performance begins to decrease again. Based on the Figure 20, a relatively small subset size R maximizes the utilization of testing resources. Subset sizes much larger than 1000 are not feasible. Unless a very large number of testing resources is available, in which case, the full coverage subset of size $R_{\min}$ might be feasible. Otherwise, resources should be allocated elsewhere.

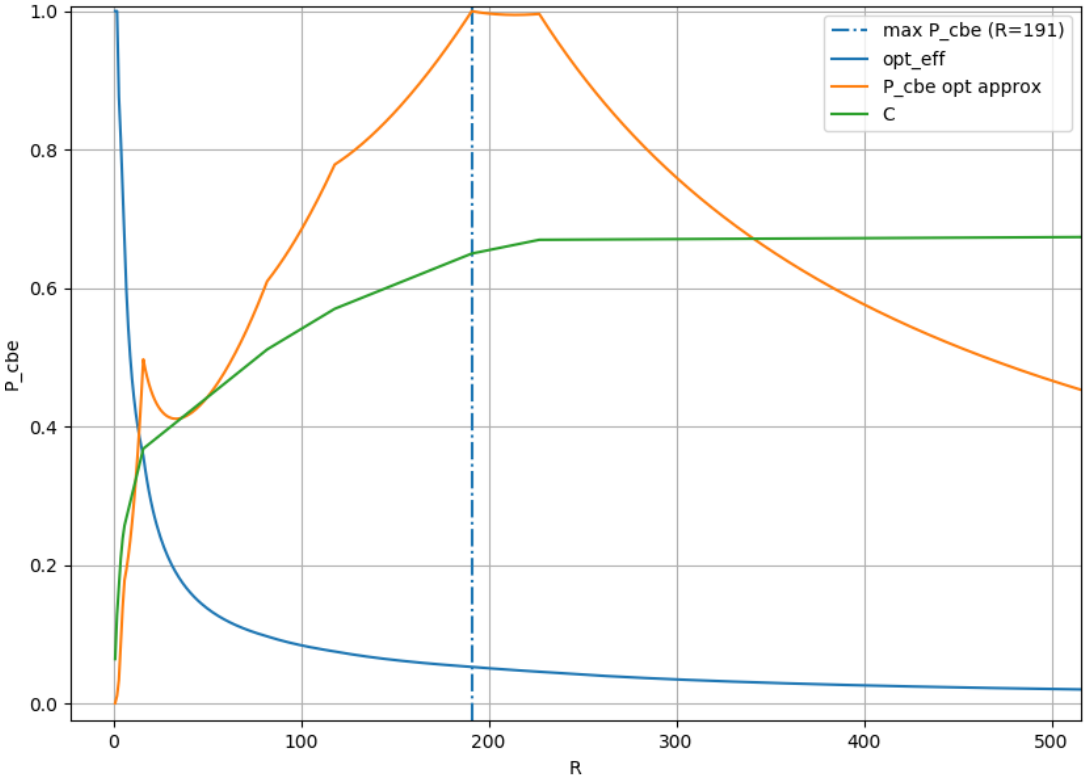

Figure 21. Zoomed graph of the approximation of the performance of an optimal subset in proximity of the maximum absolute performance, P_{cbe} with normalized values for P_{cbe} and $\varepsilon_{\mathrm{opt}}$.

Closer inspection of performance, from Figure 21, reveals that there is a local peak at R=16, which is the point where all input values of all input parameters can be included. At low R, low coverage significantly attenuates the absolute performance due to the hardness factor h. Initially, the efficiency of the subset is at maximum, because IIC, OC, and C-NIC coverages each increase the overall coverage, which is enough to compensate the small coverage. However, the efficiency decreases after there are no missing input values left, decreasing also the performance because coverage cannot no longer compensate.

At R=30, performance begins to increase again because the coverage keeps increasing at sufficient rate compared to the degrade in the efficiency as there are still many missing outputs and C-NIC combinations to be included. The performance continues to increase until no large quantities of missing C-NIC combinations can be found, from a single combination. This occurs at R=191, in which the maximum performance is achieved. After R=191, the performance does not immediately start to decrease as the coverage keeps increasing at a moderate rate as there are still C-NIC combinations to be found. After R=227, coverage increase becomes more conservative, as all C-NIC combinations have been included, thus, only output combinations are missing. It is known that at most one new output combination is possible to be included for each added combination, which explains the slow increase in the coverage when only outputs are missing. The achieved absolute performances of the tested subsets are summarized in Table 29.

Table 29. Performance results for optimized subset sizes

Subset size R	Efficiency	Coverage	Performance P_{cbe} x 10000
10	0.02922	0.2922	1.0292
100	0.005249	0.5249	2.7220
191	0.003230	0.6170	3.5831
1000	0.0006799	0.6799	1.1565
10000	0.00008032	0.8032	0.2937

7 RESULTS

Results are divided into two sections: algorithm performance results, and subset performance results. The algorithm performance results present the achieved coverage, time taken, and the overall performance of all used algorithms. The subset performance results are summarized by comparing them to the approximated optimal value.
All the algorithms were executed as a python script from the command prompt of a Windows 10 computer, using a 3.2 GHz Intel i5-6500 processor. Execution time was measured by using python's built-in time module by storing the current system time in a variable just before the executed algorithm started, and again when the algorithm finished. In this thesis, the execution time of an algorithm is defined to be the time difference of the two stored start and stop time values.

7.1 Algorithm performance results

Table 30 presents achieved coverage and the time taken for each of the tested resource limits, for all algorithms. Best algorithms with respect to achieved coverage for each of the tested resource limits are presented in Table 31. Table 32 shows the fastest algorithm for each of the used resource limits. Table 33, illustrates the overall performance of each algorithm. Overall performance is measured as average coverage per minute ratio over all the scenarios, i.e. speed, not the performance, of the algorithms is compared.

Table 30. Achieved overall coverage and required time with different algorithms

Algorithm	Resource limit									
	10		100		191		1000		10000	
	C (%)	Time	C (%)	Time	C (%)	Time	C (%)	Time	C (%)	Time
R-SGD-1	29.22	36 min	52.49	7 h	61.01	11 h	66.24	7 h	77.80	8 h
R-SGD-2	29.22	40 min	51.72	9 h	61.70	13 h	66.35	8 h	78.31	6 h
SGD-1	29.22	36 min	42.62	23 min	49.97	38 min	65.51	18 min	77.81	13 min
SGD-2	29.22	40 min	39.86	31 min	45.97	32 min	60.70	28 min	78.82	18 min
GA	25.74	16 min	47.31	45 min	58.33	70 min	67.99	79 min	80.32	80 min
WGA	25.74	16 min	47.31	45 min	58.87	72 min	67.21	81 min	79.54	82 min
R-SGD-1-GA	25.74	20 min	48.44	10 min	59.24	11 min	66.78	9 min	78.05	17 min
R-SGD-2-GA	25.74	25 min	40.01	22 min	51.52	23 min	67.19	15 min	79.12	18 min
R-SGD-1-WGA	25.74	20 min	49.06	10 min	58.83	11 min	66.50	9 min	78.16	17 min
R-SGD-2-WGA	25.74	25 min	40.01	22 min	51.31	23 min	66.60	15 min	78.94	18 min

Table 31. Best achieved coverage and respective algorithm with varying resource limits

Resource limit	Overall Coverage	Algorithm
10	29.22%	R-SGD-1
100	52.49%	R-SGD-1
191	61.70%	R-SGD-2
1000	67.99%	GA
10000	80.32%	GA

Table 32. Fastest algorithms with varying resource limits

Resource limit	Time	Algorithm
10	16 min	GA
100	10 min	R-SGD-1-WGA
191	11 min	R-SGD-1-GA
1000	9 min	R-SGD-1-GA
10000	13 min	SGD-1

Table 33. Overall algorithm performance from fastest to slowest

Algorithm	Overall performance ($\frac{1}{5}\sum_{s=1}^{5}\frac{C_{\text{overall},s}}{t_s}$) [% / min]
R-SGD-1-WGA	4.7055
R-SGD-1-GA	4.7055
R-SGD-2-GA	2.7926
R-SGD-2-WGA	2.7809
SGD-1	2.7209
SGD-2	1.9999
GA	1.0716
WGA	1.0555
R-SGD-1	0.2698
R-SGD-2	0.2522

7.2 Subset performance results

Figure 22 presents the best subset performance result for each of the tested resource limits compared to the approximated optimal performance. Figure 23 presents zoomed plot in the proximity of the maximum optimal performance.

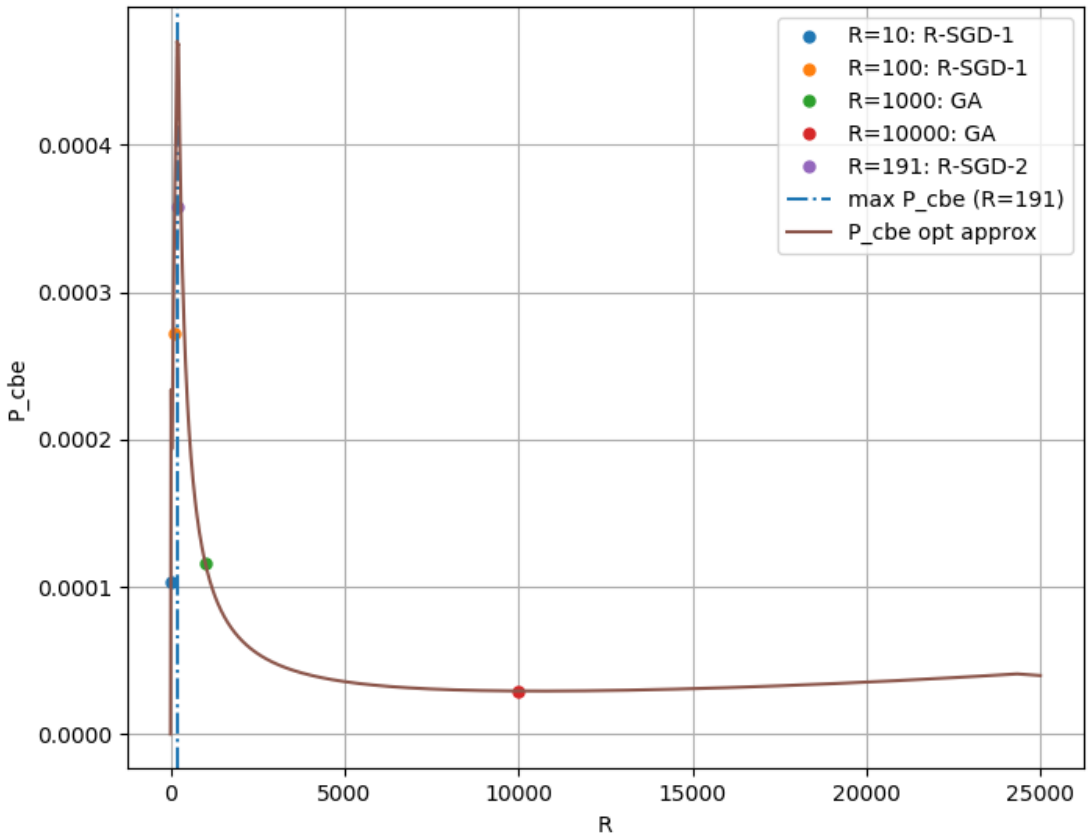

Figure 22. Subset performances of best results compared to approximated optimal performance for *R* range 0…25000.

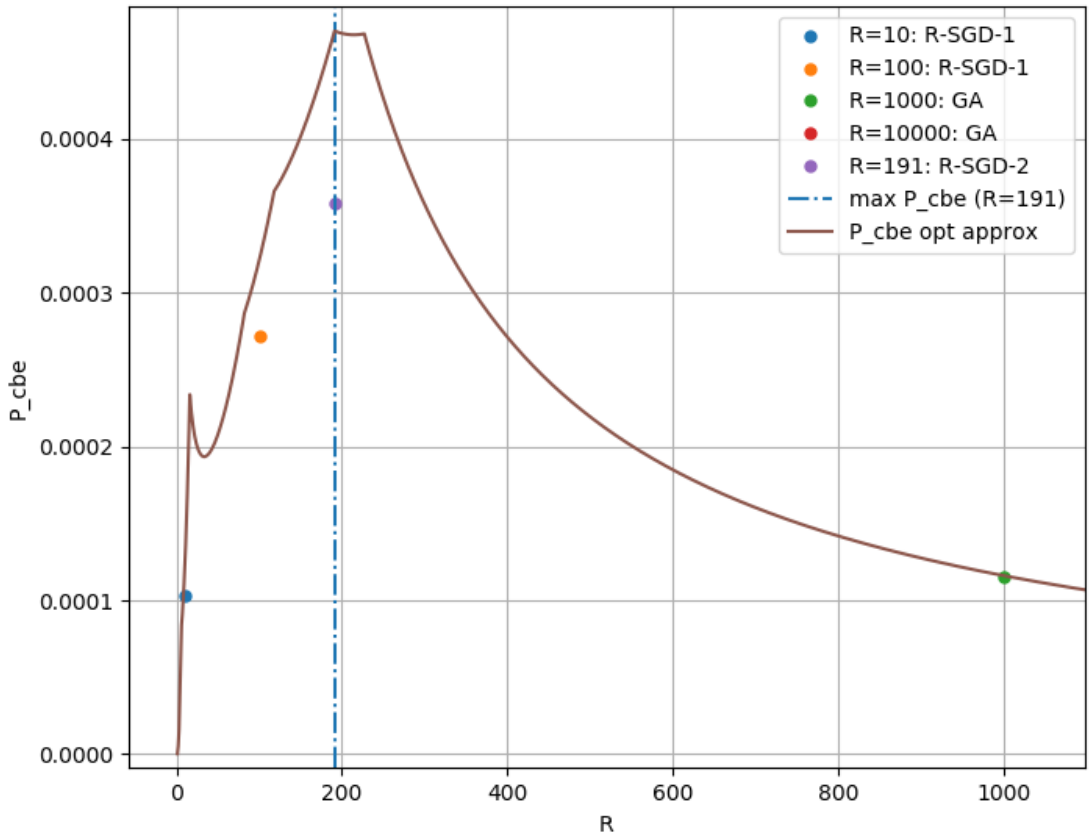

Figure 23. Zoomed view of the interesting part of the subset performance results.

8 DISCUSSION

This thesis provided tools to help to decide the parameters for testing, when there are a large number of possible combinations, and some algorithm options to be used, combined, and improved, for optimal subset search. Based on the study done in this thesis, C-NIC decomposition has the potential to significantly reduce the number of required combinations to achieve adequate testing coverage for an equation. Instead of considering all input combinations, the system state combinations can be considered, for a better utilization of the test resources.

Introduced performance metric, P_{cbe}, is one way to measure utilization of the testing resources. The definition of performance required definition of: efficiency, coverage, and a hardness factor. Definition of overall coverage used in this thesis was an unweighed average of three coverage metrics. Definition was used as an initial strategy, but the presented definition is not the best possible way to describe overall coverage level achieved with spent testing resources. The difficulty to achieve the different coverage metrics used varies which means that the coverage metric that is easiest to achieve, dominates the overall coverage. Instead, a weight factors could be used once there is better comprehension about the behavior of the three coverage components. The weights could instead be chosen so that each new element, despite the coverage metric, increases the overall coverage by a constant amount.

The introduced optimal efficiency approximation was used as reference point for relative efficiency and performance. It would be interesting to determine the accuracy of the approximation for larger subsets e.g. by performing a brute force search algorithm for a large range of resource limits and comparing the result to the approximation. Approximated value in efficiency means that everything it is used for, is also an approximation, e.g. the relative performance.

Brute force search achieves an optimal solution, but the execution time is not feasible for subset sizes much larger than 100. The proposed SGD method significantly reduced the execution time of brute force search, with the cost of losing the optimality. Classical GA and WGA were both good options for approximating the value of the $R_{\min}$. WGA in this case performed slightly better than GA. It was noticed that classical GA performance is very close to the approximated optimal solution for a large subset size. For smaller subset sizes, R-SGD resulted in the best performance among the tested algorithms. A single algorithm that could find close to optimal solution for each subset size R could not be found, i.e. the choice of algorithms depends on the available resources.

As a future work, the properties of C-NIC decomposition could be studied further and whether the result generalizes. The feasibility of a case could be studied where the GA is performed for a large CS first, after which the R-SGD is used to optimize the remaining combinations with respect to overall coverage. In addition, it could be studied if there exists way to guarantee optimality for SGD with e.g. a better sub-group division and resource allocation logic.

9 SUMMARY

The purpose of this thesis was to provide aid in deciding how to allocate limited testing resources. Limited testing resources are one of the common challenges in testing due to complex systems having a very large number of possible combinations to test. LTE was chosen as a context technology due to being sufficiently complex, widely used, and well specified technology.

A way to simplify complex equations by considering only the equation states, denoted as C-NIC decomposition, was proposed. It turned out that some equations can be simplified by dividing equation into sections, while preserving the information of the original equation among the terms. The number of state combinations might be much less than the number of input combinations, which can be exploited when deciding combinations for testing.

A metric to describe the level of test resource utilization, subset performance P_{cbe}, was introduced to be used as a target metric for the optimization. The definition of performance required the definition of: efficiency, coverage, and a hardness factor. Efficiency measured the achieved coverage per resource ratio, coverage denoted the ratio of covered and uncovered elements, and hardness factor described the complexity of the problem. An approximation of optimal efficiency was introduced to be used as a reference for the relative efficiency.

Performance was divided into absolute and relative performance. Absolute performance was a scalar number that could be used to compare the performances of the different subsets. Relative performance was a measure that described the performance of a subset to the best version of itself.

A 3GPP equation used to define PDSCH NB hopping pattern for an eMTC device was selected as a target equation, for which the search of a subset of combinations that would maximize the test resource utilization was attempted.

Finding a full coverage subset with minimum resources is so called set cover problem, for which there exists numerous algorithms, one being GA. The GA was selected for this thesis due to its simplicity and lightness. Variants of GA, WGA, and BGA were also studied. WGA turned out to be marginally better compared to GA. BGA was not feasible due to too large problem size.

With limited resources available, brute force approach is optimal. However, the brute force search could not be executed in reasonable time for subset sizes much larger than 100. Brute force search was modified into smaller sub-problems with SGD, which brought the execution time within a reasonable range. However, optimality was lost in the process because the formation of completely independent sub-groups was not possible for the chosen equation.

SGD was improved by adding the information of already included elements for the sub-groups to optimize the sub-groups with respect to whole CS, and not to maximize only the coverage of the currently optimized sub-group. Improved SGD was denoted as R-SGD. R-SGD took longer to execute but the execution time remained within reasonable range.

It turned out that the R-SGD performed well for small subset sizes, whereas GA worked better for larger subsets. The performance differences inspired a trial for a combination of R-SGD and GA algorithm, which did not increase coverage further for any subset size. Instead, the overall speed of combined algorithms was better across the tested subset sizes.

Subset performance results were compared to the approximated optimal performances and were strictly sub-optimal. Approximation error for smaller subset sizes was moderately large but quite accurate for larger subset sizes. Based on introduced performance approximation, a subset of size 191 achieved the maximum level of test resource utilization, which also achieved the highest performance of the tested set sizes i.e. 10, 100, 191, 1 000, and 10 000.

This thesis produced a way to simplify complex equations, various algorithm options for subset search, and a base for future study regarding the C-NIC decomposition, and SGD method, and inspired many algorithm ideas to be tested.

10 REFERENCES

[1] International Software Testing Qualifications Board (2018) Foundation Level Syllabys, pp. 16-17. URL: https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html (Last Accessed 5.6.2019)

[2] International Software Testing Qualifications Board (2018) Foundation Level Syllabys, page 71. URL: https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html (Last Accessed 5.6.2019)

[3] GSMA IoT deployment map. URL: https://www.gsma.com/iot/deployment-map/ (Last Accessed: 14.5.2019)

[4] Tse D; Viswanath P (2004) Fundamentals of Wireless Communication, pp 143-146.

[5] Rahnema M; Dryjanski, M (2017) From LTE to LTE-Advanced Pro and 5G, pp 1-3.

[6] (Standard) Oxford Living Dictionaries. URL: https://en.oxforddictionaries.com/definition/standard (Last Accessed: 11.03.2019)

[7] About 3GPP (2018) 3GPP official webpage URL: http://www.3gpp.org/about-3gpp

[8] (Specification) Oxford Living Dictionaries. URL: https://en.oxforddictionaries.com/definition/specification (Last Accessed: 11.03.2019)

[9] Rahnema M; Dryjanski, M (2017) From LTE to LTE-Advanced Pro and 5G, page 5.

[10] 3GPP TS 36.201 – V13.3.0 – Evolved Universal Terrestrial Radio Access (E-UTRA); LTE physical layer; General description.

[11] Chang R.W (1966) Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission. URL: https://ieeexplore-ieee-org.pc124152.oulu.fi:9443/stamp/stamp.jsp?tp=&arnumber=6769442 (Last Accessed 4.6.2019)

[12] 3GPP TS 36.211 - V13.10.0 – Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation.

[13] Andréasson N; Evgrafov A; Patriksson M (2005) An Introduction to Optimization: Foundations and Fundamental Algorithms, pp. 12-14. URL: http://apmath.spbu.ru/cnsa/pdf/monograf/Andreasson,%20Evgrafov,%20Patriksson.%20An%20introduction%20to%20continuous%20optimization.pdf (Last Accessed: 3.6.2019)

[14] Boyd S; Vandenberghe L (2004) Convex Optimization, p 71. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf (Last Accessed: 23.4.2019)

[15] Cormen T.H, Leiserson C.E , Rivest R.L, Stein C (2001) Introduction to Algorithms – 2nd Edition

[16] Geng L (2015) An implementation of harmony search algorithm to the set covering problem. 11th Internal Conference on Natural Computation (ICNC). DOI: 10.1109/ICNC.2015.7377990

[17] Crawford B, Castro C, Monfroy E (2009) A New ACO Transition Rule for Set Partitioning and Covering Problems; 2009 International Conference of Soft Computing and Pattern Recognition. DOI: 10.1109/SoCPaR.2009.89

[18] Yun-long L, Xiao-min H, Jun Z (2009) A new genetic algorithm for SET k-cover problem in wireless sensor networks; 2009 IEEE International Conference on Systems, Man and Cybernetics. DOI: 10.1109/ICSMC.2009.5346281

[19] Wen-Chih H, Cheng-Yan K, Jorng-Tzong H (1994) A genetic algorithm for set covering problems; Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence. DOI: 10.1109/ICEC.1994.349997

[20] You-Lian Z, De-Ming L (2007) Hybrid Niche Genetic Algorithm for Set Covering Problem; 2007 International Conference on Machine Learning and Cybernetics. DOI: 10.1109/ICMLC.2007.4370290

[21] D.P Chandu (2015) Improved Greedy Algorithm for Set Covering Problem URL: https://arxiv.org/ftp/arxiv/papers/1506/1506.04220.pdf (Last accessed: 29.3.2019)

[22] Kuhn R.D, Kacker R.N, Lei Y; (2016) Measuring and Specifying Combinatorial Coverage of Test Input Configurations URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5267492/ (Last accessed: 23.8.2019)

[23] 3GPP TS 36.331 - V13.13.0 – Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification

[24] 3GPP TS 36.213 - V13.13.0 – Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures

11 APPENDICES

Appendix 1 PDSCH hopping combination calculator

```
import os

num_of_combinations = 0
for i0 in range(10240):
    os.system("cls")
    print("Calculating: ", round(100 * i0 / 10240, 2), "%")
    for N_DL_NB in [1, 2, 4, 8, 12, 16]:
        for n_i0_NB in range(1, N_DL_NB):
            for f_DL_NBhop in range(1, N_DL_NB):
                for N_chDL_NBhop in [2, 4]:
                    if (N_chDL_NBhop <= N_DL_NB):
                        for N_chDL_NB in [1, 2, 4, 8]:
                            for N_PDSCH_abs in [1, 2, 4, 8, 16, 32]:
                                num_of_combinations = num_of_combinations + 1

print(num_of_combinations)
```

Appendix 2 Full C-NIC coverage combination calculator

```
import os
import math

valid_combinations = []
num_of_combinations = 0
for i0 in range(16):
   os.system("cls")
   print("Calculating: ", round(100 * i0 / 16, 2), "%")
   for N_DL_NB in [1, 2, 4, 8, 12, 16]:
      for n_i0_NB in range(1, N_DL_NB+1):
         for f_DL_NBhop in range(1, N_DL_NB+1):
            for N_chDL_NBhop in [2, 4]:
               if (N_chDL_NBhop <= N_DL_NB):
                  for N_chDL_NB in [1, 2, 4, 8]:
                     for N_PDSCH_abs in [1, 2, 4, 8, 16, 32]:
                        for N_PDSCH_val in range(1, N_PDSCH_abs+1):
                           i = i0 + N_PDSCH_abs - 1
                           j0 = math.floor(i0/N_chDL_NB)
                           T11 = math.floor((i/N_chDL_NB) - j0)
                           T12 = N_chDL_NBhop
                           T1 = T11 % T12
                           T21 = n_i0_NB + T1*f_DL_NBhop
                           T22 = N_DL_NB
                           num_of_combinations = num_of_combinations + 1
                           if [T11, T12, T21, T22] not in valid_combinations:
                              valid_combinations.append([T11, T12, T21, T22])

print(num_of_combinations)
print("Number of C-NIC combinations: ", len(valid_combinations))
```

Appendix 3 Output pattern calculator

```
import math
unique_patterns = []

for i0 in range(8):
    for N_chDL_NBhop in [2, 4]:
        for N_chDL_NB in [1, 2, 4, 8]:
            for N_DL_NB in [2, 4, 8, 12, 16]:
                for n_i0_NB in range(1, N_DL_NB+1):
                    for f_DL_NBhop in range(1, N_DL_NB+1):
                        if(N_chDL_NBhop <= N_DL_NB):
                            for N_PDSCH_abs_val in [1, 2, 4, 8, 16, 32]:
                                pattern = []
                                for N_PDSCH_abs in range(1, N_PDSCH_abs_val+1):
                                    i = i0 + N_PDSCH_abs - 1
                                    j0 = math.floor(i0/N_chDL_NB)
                                    T11 = math.floor(i/N_chDL_NB) - j0
                                    T12 = N_chDL_NBhop
                                    T1 = T11 % T12
                                    T21 = n_i0_NB + T1*f_DL_NBhop
                                    T22 = N_DL_NB
                                    output = T21%T22
                                    pattern.append(output)
                                if pattern not in unique_patterns:
                                    unique_patterns.append(pattern)

print(len(unique_patterns))
```

Appendix 4 IIC coverage calculator

```
# Handled Input Values
HIV = [ [], [], [], [], [], [], []]
# All Input Values
AIV = [ [2, 4, 8, 12, 16],                                  # N_DL_NB
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], # n_i0_NB
      [1, 2, 4, 8],                                         # N_ch_DL_NB
      [2, 4],                                               # N_ch_DL_NB_hop
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], # f_DL_NB_hop
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], # i0
      [1, 2, 4, 8, 16, 32]]                                 # N_PDSCH_abs
# All Handled Flags
AHF = [0, 0, 0, 0, 0, 0, 0]
# Previous Index
PI  = [-1, -1, -1, -1, -1, -1, -1]

combinations = []
combination = []
i = 0
skip = 0

while(sum(AHF) != len(AIV)):
    if(i == len(AIV)):
        if not(skip):
            combinations.append(combination)
            for index in range(len(combination)):
                if(combination[index] not in HIV[index]):
                    HIV[index].append(combination[index])
                if (sorted(HIV[index]) == AIV[index]):
                    AHF[index] = 1
        skip = 0
        combination = []
        i = 0
        continue
    else:
        if(AHF[i] == 1):
            if i in [1, 3, 4]:
                j = 1
                while(j < len(HIV[i]) + 1):
                    if (HIV[i][(PI[i]+j)%len(HIV[i])] <= combination[0]):
                        combination.append(HIV[i][(PI[i]+j)%len(HIV[i])])
                        PI[i] = PI[i] + 1
                        i = i + 1
                        break
                    else:
                        j = j + 1
            else:
                combination.append(HIV[i][(PI[i]+1)%len(HIV[i])])
```

```
                PI[i] = PI[i] + 1
                i = i + 1
            else:
                n = 0
                while(n < len(AIV[i])):
                    if(AIV[i][n] in HIV[i]):
                        n = n + 1
                    else:
                        if i in [1, 3, 4]:
                            best_index = -1
                            best_offset = 255
                            for m in range(len(AIV[i])):
                                offset = combination[0] - AIV[i][m]
                                if (AIV[i][m] not in HIV[i]):
                                    if(offset >= 0):
                                        if(offset < best_offset):
                                            best_offset = offset
                                            best_index = m
                            if (best_index != -1):
                                combination.append(AIV[i][best_index])
                                i = i + 1
                            else:
                                i = len(AIV)
                                skip = 1
                                break
                        else:
                            combination.append(AIV[i][n])
                            i = i + 1
                            break

    for comb in combinations:
        print(comb)

    sum = 0
    for hiv, aiv in zip(HIV, AIV):
        individual_coverage = len(hiv) / len(aiv)
        sum = sum + individual_coverage

    iic_coverage = sum / len(AIV)
    print("\nIIC COVERAGE", 100 * iic_coverage, "% WITH: ", len(combinations), "
COMBINATIONS")
```

Appendix 5 Number of combinations with restricted i_0 range.

```
import os
import math

num_of_combinations = 0
for i0 in range(16):
   os.system("cls")
   print("Calculating: ", round(100 * (i0+1) / 16, 2), "%")
   for N_DL_NB in [2, 4, 8, 12, 16]:
      for n_i0_NB in range(1, N_DL_NB+1):
         for f_DL_NBhop in range(1, N_DL_NB+1):
            for N_chDL_NBhop in [2, 4]:
               if (N_chDL_NBhop <= N_DL_NB):
                  for N_chDL_NB in [1, 2, 4, 8]:
                     for N_PDSCH_abs in [1, 2, 4, 8, 16, 32]:
                        num_of_combinations = num_of_combinations + 1

print(num_of_combinations)
```